تعلم
لغة النمذجة الموحدة ٢,٠



د/ خالد سعيد خليل
أستاذ علوم الحاسب المساعد
جامعة الملك فيصل



مركز الترجمة والتأليف والنشر

تعلم لغة النمذجة الموحدة ٢,٠

تأليف

روس مايل وكيم هاملتن

ترجمة

الدكتور/ خالد سعيد خليل

أستاذ علوم الحاسب المساعد

كلية إدارة الأعمال – جامعة الملك فيصل بالأحساء

١٤٣٢هـ/٢٠١١م

ⓒ جامعة الملك فيصل، ١٤٣٢هـ
فهرسة مكتبة الملك فهد الوطنية أثناء النشر
هاملتن. روس مايل
تعلم لغة النمذجة الموحدة ٢,٠، روس مايل وكيم هاملتن؛
خالد سعيد خليل – الأحساء، ١٤٣٢هـ.
٤٤٨ص؛ ١٧×٢٤سم
ردمك: ٠- ٠٩٧- ٠٨- ٩٩٦٠- ٩٧٨
١- الحواسيب ٢- تصميم النظم (حواسيب) أ- خليل، خالد سعيد
ديوي ٠٠٤,٢ ١٤٣٢/٨٢٥٤

رقم الإيداع : ١٤٣٢/٨٢٥٤
ردمك : ٠- ٠٩٧- ٠٨- ٩٩٦٠- ٩٧٨

العنوان الأصلي للكتاب

LEARNING UML 2.0

By
Russ Miles and Kim Hamilton

بسم الله الرحمن الرحيم

قَالَ تَعَالَى: أَعُوذُ بِاللَّهِ مِنَ الشَّيْطَانِ الرَّجِيمِ ﴿ يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوٓاْ إِذَا
قِيلَ لَكُمْ تَفَسَّحُواْ فِي ٱلْمَجَٰلِسِ فَٱفْسَحُواْ يَفْسَحِ ٱللَّهُ لَكُمْۖ وَإِذَا
قِيلَ ٱنشُزُواْ فَٱنشُزُواْ يَرْفَعِ ٱللَّهُ ٱلَّذِينَ ءَامَنُواْ مِنكُمْ وَٱلَّذِينَ أُوتُواْ
ٱلْعِلْمَ دَرَجَٰتٍۚ وَٱللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ ﴿١١﴾ ﴾ المجادلة: ١١

إهداء

أهدي هذا العمل المتواضع إلى أمي و أبي الحبيبين، وإلى زوجتي رفيقة عمري، وإلى زهرة حياتي أبنائي محمد وعمر.

كما أهدي هذا العمل إلى روح الشهيد رئيس الوزراء اللبناني السابق رفيق الحريري، الذي أتاح لي فرصة متابعة دراستي الجامعية في أرقى الجامعات الفرنسية.

وأتقدم إلى أصحاب السعادة الدكتور إبراهيم التركي والدكتور أحمد بن عبد الله الشعيبي بأسمى آيات الشكر والعرفان لما قدماه لي من دعم وتشجيع وعلى كل التسهيلات التي وفراها لنشر هذا الكتاب.

كما أتقدم بالشكر والعرفان إلى كل زملائي الكرام في كلية إدارة الأعمال وكلية علوم الحاسب وتقنية المعلومات بجامعة الملك فيصل، لتشجيعهم لي وتقديم نصائحهم الثمينة لإتمام هذا الكتاب على أكمل وجه.

مقدمة المؤلف

إن لغة النمذجة الموحدة هي أداة أو وسيلة قياسية لنمذجة الأنظمة وبخاصة البرمجية. وإذا كنت تعمل على نظام معقد أكثر من النظام البسيط الذي يستعمل عادة كأول تطبيق برمجي يُظْهِر الرسالة "مرحباً بالعام"، فمن الضروري امتلاك مهارة استعمال لغة النمذجة الموحدة، لذلك تم تقديم هذا الكتاب "تعلّم لغة النمذجة الموحدة ٢،٠".

ويهدف هذا الكتاب إلى التعمق في مفاهيم لغة النمذجة الموحدة ومخططاتها بشكل سريع وسهل، وعملي. ويقوم هذا الكتاب بتزويدك بالأدوات اللازمة لاستعمال لغة النمذجة الموحدة بفعالية في تصميم الأنظمة وإنجازها، ونشرها، وذلك من خلال مجموعة شاملة من الدروس عن مختلف أنواع مخططات لغة النمذجة الموحدة. ويغطي الكتاب المواضيع التالية:

- ملخص مختصر عن مدى فائدة لغة النمذجة الموحدة في نمذجة الأنظمة.
- كيفية أسر المتطلبات عالية المستوى في النموذج لضمان استيفاء النظام لمتطلبات مستخدميه.
- كيفية نمذجة الأجزاء المؤلفة للنظام.
- كيفية نمذجة سلوك أجزاء النظام عند تشغيلها وتفاعلها.

- كيفية الانتقال من النموذج إلى العالم الواقعي من خلال معرفة كيفية نشر النظام.
- كيفية إنشاء المظاهر profiles المكيّفة في لغة النمذجة الموحدة للتمكن من نمذجة الأنظمة في المجالات المختلفة بشكل دقيق.

الجمهور المستهدف

يتوجه هذا كتاب إلى أي شخص مهتم بتعلم لغة النمذجة الموحدة، ولكن من المفيد معرفة القليل عن التصميم الكائني التوجه و الإلمام بأساسيات لغة جافا. وعلى أية حال، فإذا كانت خبرتك قليلة مع التوجه الكائني، فسيقوم الكتاب بتطوير معرفتك بمفاهيم هذا التوجه وتوسيعها، وسيمنحك مجموعة شاملة من الأدوات للعمل مع لغة النمذجة الموحدة.

مع أنه تمت تهيئة هذا الكتاب لمرافقتك في مختلف الموضوعات في رحلة تعلم لغة النمذجة الموحدة، فإن بعض هذه المواضيع، (مثل حالات الاستخدام و مخططات النشاط) بسيطة ومفهومة تلقائياً مما يعني أنه بإمكانك الغوص فيها مباشرة.

حول هذا الكتاب

يهدف الكتاب إلى الإجابة عن الأسئلة "ماذا"، و "كيف"، و "لماذا يجب الاهتمام؟" بكل جانب من جوانب لغة النمذجة الموحدة؟ ويختار كل فصل في الكتاب موضوعاً واحداً من عناصر لغة النمذجة الموحدة حيث يتم توضيحه بالارتكاز على تلك الأسئلة.

بما أن كل القراء ليسوا جديدين على لغة النمذجة الموحدة، فإنه يوجد مساران رئيسان عبر هذا الكتاب. فإذا كنت جديداً على موضوع

لغة النمذجة الموحدة وتريد الحصول على ملخص يوضح سيرة لغة النمذجة الموحدة، فيجب عليك البدء بالفصل الأول. لكن إذا أردت الغوص في المواضيع بسرعة، فيمكنك تخطي فصل المقدمة واستكشاف حالات الاستخدام مباشرة، أو الانتقال إلى الفصل الذي يعالج مخطط لغة النمذجة الموحدة الذي يهمك.

لقد تعرفت على مضمون الكتاب، فهو غير موجه لأداة رسومية محددة تستعمل في النمذجة أو للغة برمجة معينة. وعلى أية حال، لبعض هذه الأدوات أسلوبها الخاص بترميز عناصر لغة النمذجة الموحدة، ولا تدعم بعض لغات البرمجة كل ما يمكن نمذجته في لغة النمذجة الموحدة. ومن خلال الكتاب- وعندما يكون ذلك ملائماً- حاولنا الإشارة إلى بعض الانحرافات لأدوات لغة النمذجة الموحدة، أو للغات البرمجة عن اتباع المعايير التي في لغة النمذجة الموحدة.

أخيراً، وبسبب الاختلاف الكبير في طرق تطوير البرامج، فلا يركز هذا الكتاب على طريقة أو منهجية محددة، بل يركز على عملية النمذجة وتوفير دليل إرشادي حول مستويات النمذجة الملائمة التي يمكن تطبيقها ضمن سياق عملية تطوير البرامج. وبما أن هذا الكتاب يلتزم بلغة النمذجة الموحدة ٢,٠ القياسية، فهو ذو مكانة جيدة وفقاً للمنهجية التي تقوم باستعمالها.

الفرضيات المعتمدة في هذا الكتاب

ترتكز الفرضيات العامة على معرفة القارئ وخبرته بالأمور التالية:

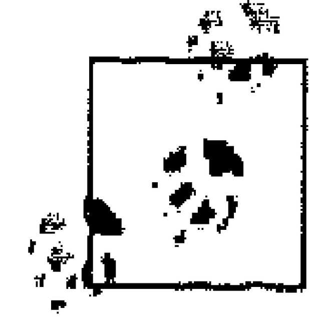
- إدارك و فهم التوجه الكائني
- معرفة لغة البرمجة جافا بخصوص بعض الأمثلة

الاعتبارات المتبعة في هذا الكتاب

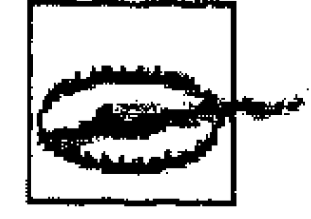
تتلخص الاعتبارات الطباعية المستعملة في هذا الكتاب في:

خط داكن:

للإشارة إلى الشروط الجديدة، أسماء الملفات، امتدادات الملفات، مسارات الأسماء، الأدلة، الخيارات، المفاتيح، المتغيرات، الخواص، الوظائف، الأنواع، الأصناف، فضاءات الأسماء، الطرق، الوحدات، الميزات، البارامترات، القيم، الكائنات، الأحداث، مدراء الحدث، الماكرو (برنامج إحلالي)، ومحتويات الملفات، أو ناتج الأوامر.



تشير هذه الأيقونة إلى نصيحة، أو اقتراح، أو ملاحظة عامة.



تشير هذه الأيقونة إلى تحذير أو احتراس.

استعمال شفرة الأمثلة

الهدف من الكتاب هو مساعدتك على إنجاز عملك. وقد تقوم باستخدام عام لشفرة هذا الكتاب في برامجك ووثائقك. ولست بحاجة إلى الاتصال بنا لطلب رخصة لهذا الاستخدام ما لم تعد إنتاج جزء مهم من الشفرة. على سبيل المثال، لست بحاجة لرخصة في حال كتابة برنامج يستعمل عدة قطع كبيرة من شفرة هذا الكتاب، ولكن عملية بيع أو توزيع قرص مدمج عن أمثلة من كتب أورايلي O'Reilly تتطلب رخصة. وإجابتك عن سؤال بذكر هذا الكتاب و الاستشهاد بشفرة الأمثلة لا

يتطلب رخصة، أما دمج كمية هامة من شفرة الأمثلة من هذا الكتاب في توثيق منتجك فهو يتطلب رخصة.

كيفية الاتصال بنا:

لقد تم تجهيز الكتاب بأمثلة دقيقة، ومتحقق منها حسب قدرة المؤلفين و المترجم. على أية حال، رغم أن UML هي لغة نمذجة قياسية، فإن أفضل الممارسات بالنسبة لاستعمالها قد تتغير مع الوقت، و ربما يكون لهذا تأثير على محتويات هذا الكتاب.

هناك صفحة ويب لهذا الكتاب حيث يمكن أن تجد الأخطاء الواردة في الكتاب الإنجليزي، وبعض الأمثلة و المعلومات الإضافية. ويمكنك الدخول إلى هذه الصفحة عبر الموقع: http://www.oreilly.com/catalog/learnuml2. للتعليق أو طلب أسئلة تقنية حول الكتاب الإنجليزي، البريد الإلكتروني: bookquestions@oreilly.com. للاتصال بالمترجم، البريد الإلكتروني: khaled.khalil.books@hotmail.com.

مقدمة المترجم

لقد أحدثت المفاهيم الكائنية التوجه نقلة نوعية وجذرية في أسلوب نمذجة الأنظمة و تمثيل الواقع، حيث أثّرت في أغلب مجالات علوم الحاسب من تحليل، وتصميم، وبرمجة الأنظمة و قواعد البيانات ومجالات عدة أخرى. وذلك لما لهذه التقنية من مفاهيم تدعم متطلبات العمل في هذه المجالات من تنظيم تقسيم العمل، والعمل ضمن مجموعات، وإعادة الاستعمال و الصيانة.

وقدمت لغة النمذجة الموحدة عملاً رائعاً في توحيد طرق النمذجة خاصة تلك المرتكزة على التوجه الكائني والمستعملة بكثرة قبل عام ١٩٩٧م. كما أنها لغة نمذجة كائنية التوجه بامتياز، حيث أخذت نقاط القوة والطرق الموجودة سابقاً وجمعتها في لغة واحدة ذات أسلوب رسومي سلس وجذاب، مما أدى إلى تجنب كل التباس وغموض ناتج عن طرق النمذجة السابقة. وقد وفرت هذه اللغة أداة عمل للفرق المشاركة في مراحل نمذجة النظم وإنجازها ونشرها.

وقامت العديد من الشركات العملاقة، في المجالات الصناعية والمعلوماتية، باعتماد هذه اللغة وإدخالها في منتجاتها سواء كانت أدوات نمذجة ورسم مخططات، أو بيئات برمجية، أو أدوات أخرى.

يشرح هذا الكتاب لغة النمذجة الموحدة ٢.٠ بأسلوب بسيط و عملي، حيث يتطرق إلى مختلف مفاهيم البرمجة كائنية التوجه من الناحية النظرية، ويبين أيضاً كيفية برمجة عناصر اللغة النمذجية باستعمال الإصدار الخامس من لغة البرمجة جافا. ويقدم الكتاب مثالاً تطبيقياً واقعياً حول المدونات حيث يرافق القارئ تدريجياً عبر فصول الكتاب.

ويتوجه هذا الكتاب إلى كل شخص مهتم بعلوم الحاسب أو نظم المعلومات أو هندسة برمجيات، أو أي تخصص آخر ذات علاقة بنمذجة الأنظمة. ويمكن أن يستفيد منه أي شخص على علاقة بالمجالات المذكورة ويرغب بتعلم لغة النمذجة الموحدة ٢.٠ أو تحديث معلوماته عنها.

لقد حاولت قدر المستطاع إيجاد التعابير الأكثر دقة للمواضيع المطروحة، لشرحها وتبسيطها وتقديمها بأسلوب مفهوم وقريب للقارئ. وقمت أيضاً بذكر المصطلحات الإنجليزية في الكتاب، بخاصة عند أول ظهور لها، كي لا يخلق التباس عند القارئ بسبب عدم توفر مصطلحات عربية مقابلة لها معتمدة وموحدة بين العاملين في هذا المجال.

أتمنى أن يستفيد من هذا الكتاب كل من يرغب بتعميق معرفته في مجال النمذجة وكل طالب علم، راجياً من المولى الكريم أن يوفقني لتقديم مزيد من العلوم إلى مجتمعنا العربي، ولإغناء المكتبة العربية بأحدث العلوم خصوصاً المتعلقة بمجال علوم الحاسب ونظم المعلومات.

الدكتور خالد سعيد خليل

المحتويات

الفصل الأول مقدمة

الفصل الثاني نمذجة المتطلبات: حالات الاستخدام

الفصل الثالث نمذجة تدفقات عمل الأنظمة: مخططات النشاط

الفصل الرابع: نمذجة الهيكل المنطقي للنظام: تقديم الأصناف ومخططات الأصناف

الفصل الخامس: نمذجة الهيكل المنطقي للنظام: مخططات الأصناف المتقدمة

الفصل السادس: نقل الأصناف إلى الحياة: مخططات الكائنات

الفصل السابع: نمذجة التفاعلات المرتبة: مخططات التتابع

الفصل الثامن: التركيز على روابط التفاعل: مخططات الاتصال

الفصل التاسع: التركيز على توقيت التفاعل: مخططات التوقيت

الفصل العاشر: إتمام وصف التفاعل: مخططات ملخّص التفاعل

الفصل الحادي عشر: نمذجة الهيكل الداخلي للصنف: الهياكل المركبة

الفصل الثاني عشر: إدارة أجزاء النظام وإعادة استعمالها: مخططات المكوّنات

الفصل الثالث عشر تنظيم النموذج: الحُزَم

الفصل الرابع عشر: نمذجة حالة الكائن: مخططات حالة الآلة

الفصل الخامس عشر: نمذجة النظام المنشور: مخططات النشر

الملاحق

أ- لغة قيود الكائن

ب- تكييف لغة النمذجة الموحدة: المظاهر

ج- لمحة تاريخية عن لغة النمذجة الموحدة

مقدمة

INTRODUCTION

إنّ لغـة النمذجـة الموحّـدة UML هـي لغـة نمذجـة قياسـية لتطـوير الأنظمة والبرمجيات. و يشكل هذا الطرح بحد ذاته حجة جيدة وحاسمة لجعل UML جزءاً من ذخيرتك البرمجية، و مع ذلك تبقى بعض الأسئلة من دون أجوبة عليها. لماذا تعتبر UML لغة موحّدة؟ وما الذي يمكنها نمذجته؟ وكيف تكون لغة بحد ذاتها؟ و لماذا عليك الاهتمام بها؟

إن تصميم الأنظمة على أي نطاق واسع نسبياً هو أمر صعب، فأي نظام يكون انطلاقاً من نظام تطبيقي مكتبي بسيط إلى نظام على نطاق مشروع مؤسسي كامل متعدد المستويات، ويمكن أن يتكون من المئات وربما الآلاف من المكوّنات المادية و البرمجية. وكيف يمكنك أنت (وفريق عملك) من تعقّب المكوّنات المطلوبة؟ وما هي وظائفها؟ و كيف تلبي متطلبات زبائنك؟ علاوة على ذلك، كيف يمكنك مشاركة تصميمك مع زملائك لضمان عمل المكونات معاً؟ هناك كثير من التفاصيل التي يمكن أن يُساء تفسيرها أو يتم نسيانها عند تطوير نظام معقّد من دون أي مساعدة. من هنا تأتي أهمية النمذجة وبالطبع لغة النمذجة الموحدة.

في تصميم الأنظمة، نقوم بالنمذجة لسبب مهم مفاده إدارة التعقيد. حيث تساعد النمذجة على رؤية التصوّر الكبير مع التركيز على

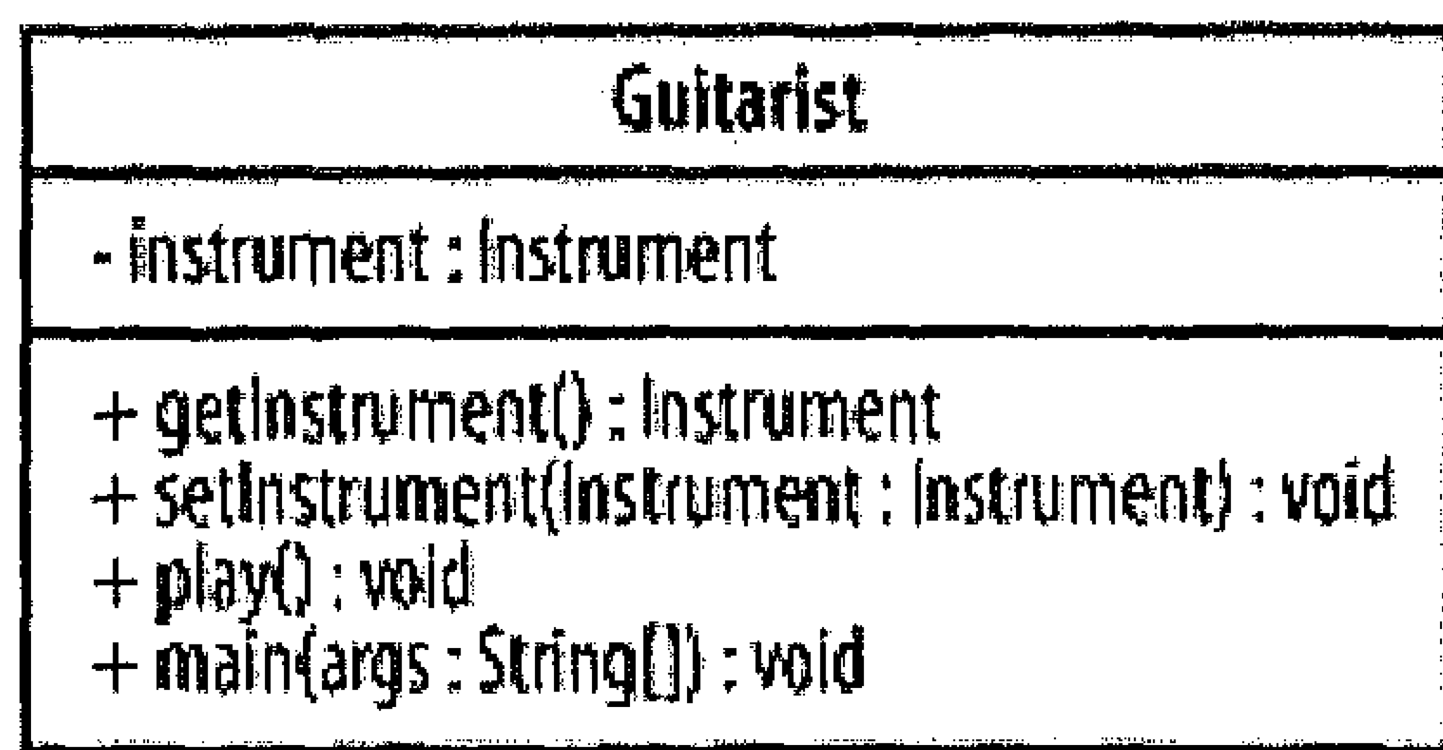
التفاصيل، حيث تسمح لك بالتركيز على أسر السمات المهمة لتصميم نظامك وتوثيقها ونقلها.

النموذج هو تجريد abstraction لمسألة واقعية. عند نمذجة نظام محدد، تستبعد أي تفاصيل غير مهمة، أو قد تكون مربكة. يكون النموذج عبارة عن تبسيط simplification للنظام الحقيقي، لذلك يسمح النموذج بأن يكون تصميم النظام و آلية عمله أمراً مفهوماً، ويمكن تقييمه وانتقاده بشكل أسرع عما كان عليك التعمّق في النظام الحقيقي نفسه. و بشكل أفضل، فمع لغة نمذجة رسمية formal ما، تكون اللغة مجرّدة لكن بالدقة التي تكون عليها لغات البرمجة. تسمح هذه الدقّة للغة ما بأن تكون مقروءة آلياً، وبالتالي يصبح بالإمكان تفسيرها، وتنفيذها، وتحويلها بين الأنظمة.

ومن أجل نمذجة نظام ما بشكل فعّال، نحتاج إلى أمر مهم جداً ألا وهو: لغة تمكننا من وصف النموذج، ومن هنا تأتي الحاجة إلى لغة النمذجة الموحدة.

١-١ ماذا يوجد في لغة النمذجة؟
What's in a Modeling Language?

يمكن أن تتكون لغة النمذجة من شبه الشفرة، أو شفرة حقيقية، أو صور، أو مخططات، أو مقاطع توصيف طويلة؛ في الحقيقة، وهي شيء بارع جداً يمكن أن يساعد في وصف النظام. وتسمى العناصر التي تؤلف لغة النمذجة ترميزات اللغة notation. يعرض الشكل (١-١) مثالاً لعنصر من عناصر ترميز لغة UML.



شكل رقم (١-١) عرض التصريح عن صنف class باستعمال ترميز UML للأصناف.



يتخلل هذا الكتاب بعض الإشارات إلى مصطلح نموذج النموذج meta-model و المظاهر profiles الخاصة بلغة UML. يتوفر في الملحق (ب) وصف أكثر شمولاً عن محتوى نموذج النموذج في لغة UML وعن سبب فائدته في النمذجة. لكن حتى الآن، فَكِّر فقط بنموذج النموذج للغة UML كأنه وصف وتعريف لمعنى كل عنصر ترميز في UML، و عنصر المظهر، مثل التكييف لهذا الوصف ليتماشى مع مجال محدّد كالأنظمة المصرفية.

على أية حال، ليس الترميز هو القضية بالكامل، فمن دون القول أن الصندوق الذي في الشكل رقم (١-١) يمثل صنفاً، فلن تعرف حتماً معنى هذا الصندوق، رغم إنه بإمكانك تخمين ماهيته. وتسمى توصيفات معاني الترميزات بدلالية semantics اللغة و يمكن أسرها في نموذج النموذج للغة.

ويمكن أن تكون لغة النمذجة أيّ شيء يتضمن ترميزاً ما (هي وسيلة للتعبير عن النموذج) ووصف لمعنى هذا الترميز (نموذج النموذج). لكن لماذا علينا التفكير باستعمال UML عندما يتوفر العديد من طرق النمذجة المختلفة، بما فيها تلك التي يمكن أن تصنعها على طريقتك الخاصة؟

لكل طريقة نمذجة ميزات وعيوب متفاوتة، و لكن تتميز لغة UML بست ميزات رئيسية:

١- إنها لغة رسمية **formal**: كل عنصر من اللغة له معنى معرّف بدقة، وبالتالي يمكن الاطمئنان بأنه عند نمذجة جزئية معيّنة من النظام فلن يتم إساءة فهمها.

٢- إنها مختصرة **concise**: تتكون اللغة برمتها من ترميزات بسيطة وصريحة.

٣- إنها شاملة **comprehensive**: إنها تصف كل السمات المهمة للنظام.

٤- إنها قابلة التوسع **scalable**: عندما يكون ذلك ضرورياً، فاللغة رسمية بدرجة كافية لإدارة مشاريع نمذجة أنظمة ضخمة، وهي تناسب أيضاً العمل على مقياس الأنظمة الأصغر مع المشاريع الصغيرة، و هذا يغنينا عن تعلم عدة طرق نمذجة مختلفة واستعمالها.

٥- إنها مبنية على الدروس المكتسبة: إن لغة UML هي نتيجة لأفضل الممارسات التطبيقية في مجتمع التوجه الكائني خلال السنوات الخمس عشرة الماضية.

٦- إنها قياسية **standard**: يسيطر على UML مجموعة معايير مفتوحة مع مساهمات نشطة من مجموعة عالمية من الباعة و الأكاديميين، التي تواجه الاعتماد على الباعة. وتضمن هذه المعايير قابلية التغيير، وقابلية العمل الجماعي للغة UML، و هذا يعني أنك غير مرتبط بمنتج ما من قبل بائع محدد.

١-١-١ الإفراط بالتفاصيل: النمذجة باستعمال الشفرة
Detail Overload: Modeling with Code

تعتبر شفرة البرامج مثالاً عن لغة نمذجة محتملة، حيث لا يستبعد أي جزء من التفاصيل. وكل سطر من الشفرة هو تفصيل لما يُتوقع أن يعمله برنامجك. يعرض المثال رقم (١-١) تعريفاً بسيطاً للصنف Guitarist في لغة جافا، و مع ذلك فيوجد كثير من التفاصيل في هذا التعريف.

المثال (١-١) تعريف صنف بسيط في جافا يحتوي على كثير من التفاصيل للتبحر فيها.

```
package org.oreilly.learningUML2.ch01.codemodel;

public class Guitarist extends Person implements MusicPlayer {

 Guitar favoriteGuitar;

 public Guitarist (String name) {
 super(name);
 }
 طريقتان محليتان لأجل الوصول إلى بيانات الصنف //
 public void setInstrument(Instrument instrument) {
 if (instrument instanceof Guitar) {
 this.favoriteGuitar = (Guitar) instrument;
 }
 else {
 System.out.println("أنا لا أعزف هذا الأمر!");
 }
 }

 public Instrument getInstrument( ) {
 return this.favoriteGuitar;
 }
 // MusicPlayer من الأفضل برمجة هذه الطريقة كما يفرضه
 public void play( ) {
 System.out.println(super.getName() + "... يقوم الآن بالعزف على القيثارة");

 if (this.favoriteGuitar != null) {
 for (int strum = 1; strum < 500; strum++) {
```

```
this.favoriteGuitar.strum( );
}
System.out.println("الانتهاء من كل المعزوفات المرهقة");
}
else {
System.out.println("لم تعطني قيثارة إلى الآن!");
}
}
// هذا القسم خاص بالبرنامج الرئيسي الذي يجب برمجته أيضاً
public static void main(String[] args) {
MusicPlayer player = new Guitarist("Russ العازف");
player.setInstrument(new Guitar("Burns Brian May Signature القيثارة"));
player.play( );
}
}
```

يعرض المثال رقم (١-١) كل المعلومات عن الصنف قيثارة Guitar، بما فيها علاقات الوراثة مع أصناف أخرى، والمتغيرات الأعضاء البيانية المرتبطة بالأصناف الأخرى، و حتى التفصيلات البرمجية للطرق نفسها.

ما الخطأ في استعمال شفرة المصدر للبرامج كنموذج لك؟ حيث تكون كل التفاصيل موجودة فيه، و كل عنصر من ترميزات اللغة له معنى بالنسبة للمترجم compiler، و مع بعض التعليقات الفعلية على مستوى الشفرة؛ مثل تعليقات التوثيق الخاصة بلغة جافا JavaDoc. ألا يكون حينئذ عندك تمثيل دقيق لبرنامج نظامك؟

الحقيقة أنك لم تنمذج فعلياً أيّ شيء سوى الشفرة الخاصة بالبرنامج. وتركز شفرة المصدر على البرنامج نفسه فقط، وتهمل بقيّة تفاصيل النظام. وبالرغم من أن شفرة المصدر هي تعريف كامل وبشكل عام واضح عمّا سيعمله البرنامج، إلا أنها لا تستطيع وحدها إخبارك ببساطة عن كيفية استعمال البرنامج ومَن سيستعمله، ولا حتى عن

كيفية نشره؛ كما ستغيب كلياً الصورة الكبرى للنظام إذا كان عندنا شفرة المصدر فقط.

بالإضافة إلى إهمال الصورة الكبرى للنظام، فإن شفرة البرنامج تعتريها مشكلة الحاجة إلى استعمال تقنيات وطرق أخرى لتوضيح النظام للآخرين. كما يجب عليك فهم شفرة المصدر لقراءتها، غير أنها تعتبر لغة مطوّري البرامج ولا يستطيع فهمها العاملون الآخرون على النظام، كالزبائن ومصممي النظام. وقد يريد هؤلاء الآخرون التركيز على المتطلبات فقط، أو ربما مشاهدة كيف تعمل مكوّنات النظام سويّاً لتحقق تلك المتطلبات. وبما أن شفرة المصدر مفرطة في تفاصيل عمل البرنامج، فهي لا تستطيع تزويدنا برؤية مجردة عالية المستوى عن النظام تناسب تلك الفئات من العاملين على النظام.

تخيّل الآن أنك طورت نظامك باستعمال تشكيلة من لغات البرمجة، مما سيزيد المشكلة سوءاً ببساطة. فليس أمراً عملياً أن تطلب من جميع العاملين على النظام تعلم تلك اللغات قبل أن يتمكنوا من فهم النظام.

أخيراً، إذا تمت نمذجة التصميم باستعمال شفرة المصدر، فإنك تخسر أيضاً عندما يتعلق الأمر بمفهوم إعادة الاستعمال، فالتصميم يكون عادة قابلاً لإعادة الاستعمال بينما لا يمكن ذلك دائماً مع شفرة المصدر. على سبيل المثال، تكون إعادة برمجة تطبيق جافا من النوع Swing (يشمل واجهات المستخدم الرسومية) باستعمال لغة HTML أو تقنية دوت نت (NET.) أسهل بكثير إذا تم نمذجة التصميم بدلاً من إجراء هندسة عكسية للشفرة (تعني الهندسة العكسية هنا استخلاص تصميم النظام من خلال شفرته المصدرية).

وتنتج كل هذه المشاكل عن حقيقة وهي أن شفرة المصدر توفر مستوىً واحداً فقط من التجريد: مستوى برمجة النظام. ولسوء الحظ هذه المشكلة الجذرية تجعل شفرة المصدر لغة فقيرة للنمذجة.

١-١-٢ الإسهاب والغموض والالتباس: النمذجة مع اللغات غير الرسمية

Verbosity, Ambiguity, Confusion: Modeling with Informal Languages

تأتي اللغات غير الرسمية في الطرف الآخر للطيف في مقابل نماذج الشفرة المصدرية الكاملة و الدقيقة. ولا يوجد ترميز معرّف رسمياً للغات غير الرسمية؛ وليست هناك قواعد صارمة لتحديد معنى ترميز معيّن على الرغم من إمكانية وجود دليل إرشادي بهذا الخصوص.

إن اللغة الطبيعية هي مثال جيد للّغات غير الرسمية، و اللغة الطبيعية - كالتي تقرؤها في هذا الكتاب - مشهورة بالالتباس في معانيها. والتعبير بدقة عن شيء ليصبح مفهوماً لأي شخص، هو تحدٍّ في أحسن الأحوال و مستحيلاً في أسوئها. إن اللغة الطبيعية مرنة ومسهبة، وهي ملائمة للمحادثة، لكنها تشكل مشكلة حقيقية عند استعمالها لنمذجة الأنظمة.

إن الوصف التالي هو نموذج في اللغة الطبيعية للمثال (١-١)، وهو مبالغ فيه قليلاً، ولكنه دقيق تقنيا:

إن عازف القيثارة Guitarist هو صنف يحتوي على ستة أعضاء: عضو ساكن static، والخمسة الباقية غير ساكنين non-static. وبما أن هذا العازف يستعمل قيثارة، فإن الصنف Guitarist يحتاج إلى مثيل instance من الصنف قيثارة Guitar؛ وعلى أية حال، بما أن الصنف Guitar

سيكون مشتركاً مع أصناف أخرى في حزمته package، فقد تم التصريح عن المتغير Guitar favorite الممثل للصنف Guitar باستعمال الخاصية افتراضي default (وهي نوع من محددات الوصول إلى أعضاء الأصناف للتمكن من الوصول المباشر إليها من داخل حزمتها).

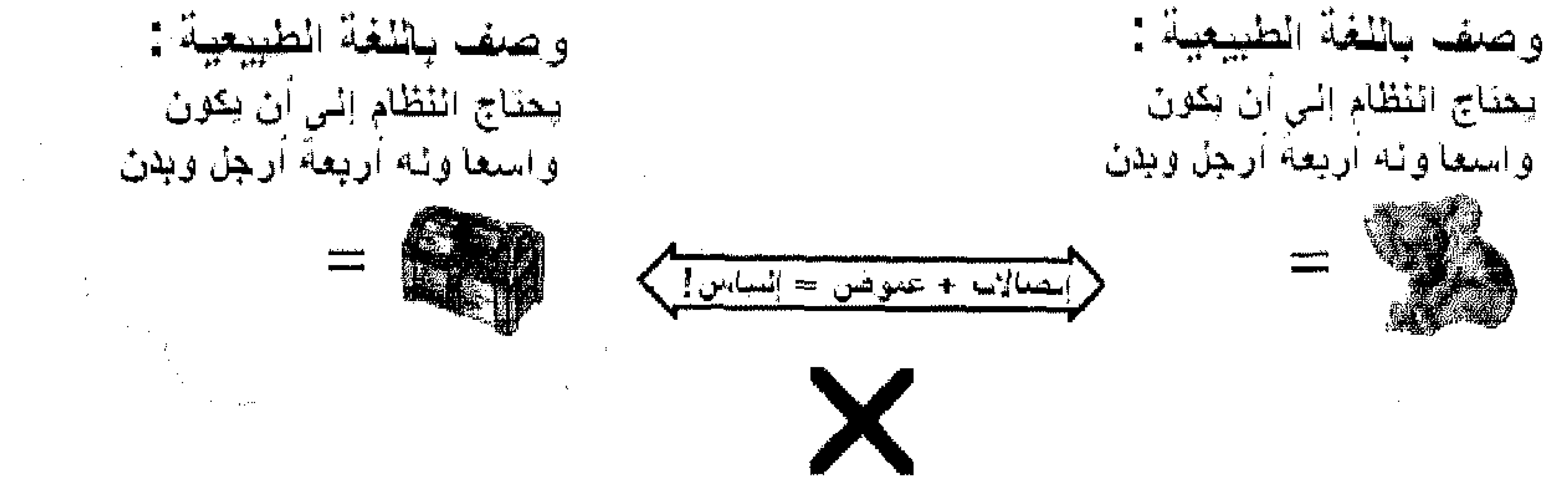
إن خمسة من الأعضاء التي داخل الصنف Guitarist هي طرق methods؛ منها أربعة غير ساكنة. وإحدى هذه الطرق الأربعة عبارة عن مشيد constructor يأخذ وسيطة argument واحدة فقط، وهي عبارة عن مثيل سلسلة رموز من الصنف String اسمها name، حيث يلغي المشيد التعريف التلقائي للمشيد الافتراضي الذي يكون من دون وسيطات.

يوفر الصنف Guitarist ثلاث طرق عادية؛ ولقد سميت الطريقة الأولى setInstrument (لتحديد آلة العزف)، حيث تأخذ وسيطة واحدة اسمها: instrument، وهي عبارة عن مثيل من الصنف: آلة عزف Instrument، وليس لها نوع إرجاع. و سميت الطريقة الثانية: getInstrument حيث إنها من دون وسيطات، إنما نوع إرجاعها هو Instrument. و سميت الطريقة الأخيرة: play، وهي مصرح عنها عملياً في الواجهة MusicPlayer التي ينجزها الصنف Guitarist. ولا تأخذ الطريقة play وسيطات، و نوع إرجاعها هو void (أي لا ترجع قيمة).

أخيراً، إن الصنف Guitarist هو أيضاً برنامج يمكن تنفيذه؛ لأنه يحتوي على طريقة تتطابق مع مواصفات جافا للطريقة الرئيسة main، المستخدمة للبرنامج الرئيسي في لغة جافا.

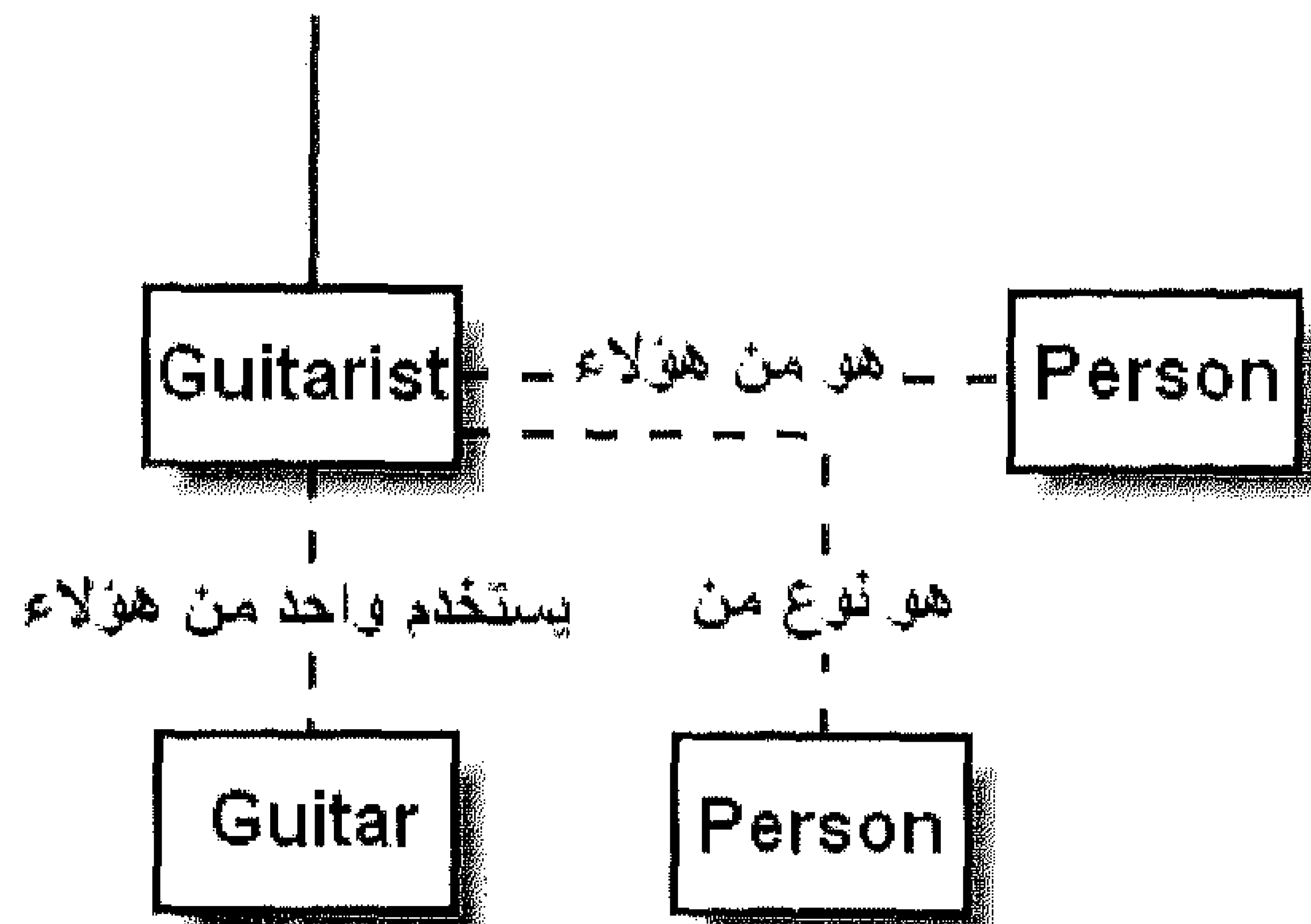
وإذا نظرت بتمعن إلى هذا التعريف، يمكنك رؤية المشاكل في كافة أنحائه، حيث إن معظمها ناتج عن الالتباس في اللغة. وينتج هذا الالتباس عندما نقوم بوصف شيء ما بشكل واضح قدر الإمكان،

ولكن يسيء الشخص الذي أوكلت إليه التصميم فهم قصدك فتقول له: "لا ، ليس هذا ما قصدت!"، انظر إلى الشكل (١-٢).



منظور مُنجز النظام   منظور مُصمّم النظام

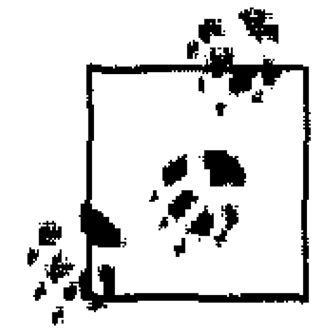
شكل رقم (١-٢) يمكن حتى لجملة بسيطة باللغة الطبيعية من أن تُفهم بشكل مختلف كلياً من قِبَل أشخاص مختلفين يعملون على النظام.



شكل رقم (١-٣) قد يكون الترميز غير الرسمي مشوشاً؛ بالرغم من وضوح ما قصدت في هذا المخطط، إلا أنه لا يمكن التأكد حقاً من معنى الترميز ما لم أخبرك به.

لا تقتصر مشاكل اللغات غير الرسمية بأي حال من الأحوال على اللغات المكتوبة فقط. فالشكل رقم (١-٣) هو مثال آخر على اللغة غير الرسمية لنفس وصف الصنف Guitarist، لكنه يأتي على شكل رسمة، حيث قمت بإنشاء هذا الترميز بنفسي. ولهذا الترميز معنى كامل بالنسبة لي، لكن يمكن إساءة فهمه بسهولة من قبل الآخرين.

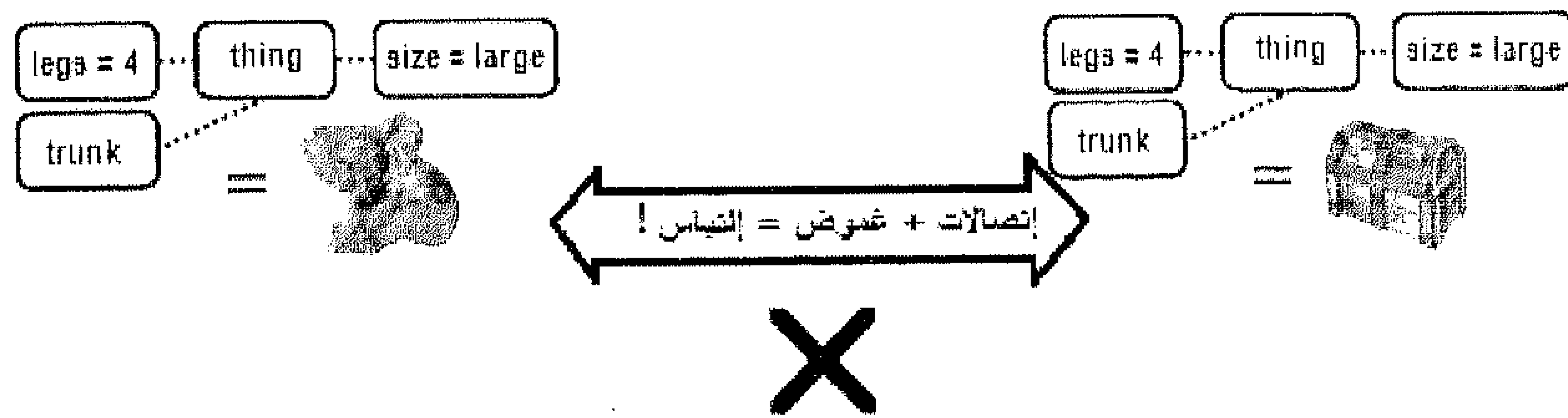
كما هو الحال مع نموذج اللغة الطبيعية، فكل التفاصيل موجودة في الشكل رقم (١-٣)، لكن من دون تعريف معنى الصناديق والارتباطات و العلامات، لا يمكن التأكد من تفسيرات الآخرين (أو من تفسيراتي!).



لذلك، لماذا الاهتمام بتلك الأمور إذا كان عند فريقك تقنية نمذجة خاصة به يستعملها منذ سنوات، و كنتم جميعاً تفهمون ما يعنيه بعضكم بعضاً؟ إذا كان عليك عرض تصميمك على المعنيين بالأمر من خارج مؤسستك، فقد يحبطون من محاولة فهم الرموز الخاصة بك، في حين كان بإمكانك استعمال ترميز قياسي يعرفونه مسبقاً. وهذا يعني أيضا أنك لست مضطراً إلى تعلم تقنية نمذجة جديدة متى ما أردت تغيير وظيفتك!

إن المشكلة الأساسية مع اللغات غير الرسمية هي عدم احتوائها على قواعد دقيقة لترميزاتها. ففي مثال اللغة الطبيعية، كانت معاني جُمَل النموذج مبهمة بسبب غموض اللغة الإنجليزية وإسهابها. ربما لم تُعانِ الصورة في الشكل رقم (١-٣) من نفس مشاكل الإسهاب، لكن من دون معرفة ما تمثله الصناديق والروابط فيها، ويكون معنى النموذج قد ترك للتخمين.

وبما أن اللغات غير الرسمية ليست دقيقة، فلا يمكن تحويلها إلى شفرة كما هو الحال مع اللغات الرسمية. ولك أن تتخيل إذا كان للشكل رقم (١-٣) مجموعة قواعد رسمية، بالتالي يمكنك توليد شفرة مصدر تنجز الأصناف عازف القيثارة Guitarist، وشخص Person، وغير ذلك. ولكن هذا مستحيل من دون فهم القواعد. ولسوء الحظ، ستعاني اللغات غير الرسمية دائماً من مشكلة الإسهاب والغموض المزدوجة، ولهذا فهي تقنية فقيرة وأحياناً خطيرة جداً لنمذجة الأنظمة، كما هو معروض في الشكل رقم (١-٤).



منظور المعنيون بالنظام مثل العميل

منظور مصمم النظام

شكل رقم (١-٤) مع الترميز غير الرسمي، تبقى مشكلة الالتباس موجودة بسبب الغموض.



رغم غموض اللغة الطبيعية بشكل خطير، فلا تزال إحدى أفضل الطرق لأسر المتطلبات، كما سنرى عندما نتعلم عن حالات الاستخدام **use cases** في الفصل الثاني.

١-١-٣ إيفاء الميزان: اللغات الرسمية

Getting the Balance Right: Formal Languages

لقد رأيت بعضاً من مخاطر استعمال لغة نمذجة فائقة التفصيل (شفرة المصدر) ولغة نمذجة فائقة الإسهاب والغموض (اللغة الطبيعية).

ولنمذجة نظام ما بشكل فعّال - مع تفادي الإسهاب و الغموض والتفاصيل غير الضرورية - فإنك تحتاج إلى لغة نمذجة رسمية.

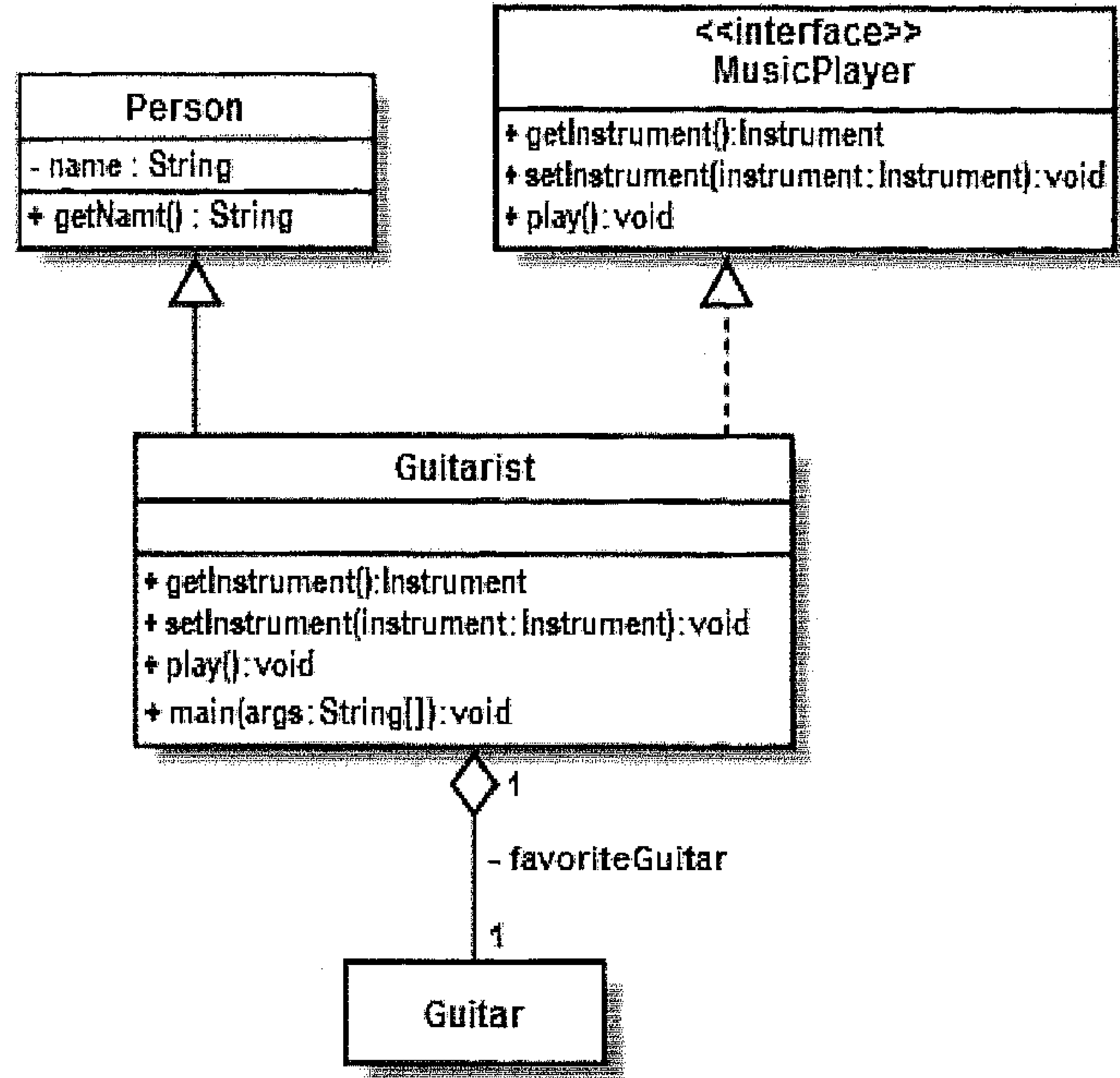
وبشكل مثالي، تضم لغة النمذجة الرسمية ترميزاً بسيطاً مما يجعلها معرّفة بإتقان. ويجب أن يتسم ترميز لغة النمذجة بكونه موجزاً بشكل كاف حتى يسهل تعلمه، ويجب أيضاً على معنى الترميز أن يكون واضح التعريف. ولغة UML ليست إلا مجرد مثال عن لغة نمذجة رسمية.

كما يعرض الشكل رقم (١-٥) كيفية التعبير عن هيكل الشفرة التي في المثال رقم (١-١) باستعمال لغة UML. في الوقت الحاضر، لا تقلق كثيراً حول الترميزات أو معناها؛ فالهدف هو استعمال مخطط UML لمجرد المقارنة بينها وبين نماذج اللغة الطبيعية واللغة التصويرية غير الرسمية المعروضة سابقاً.

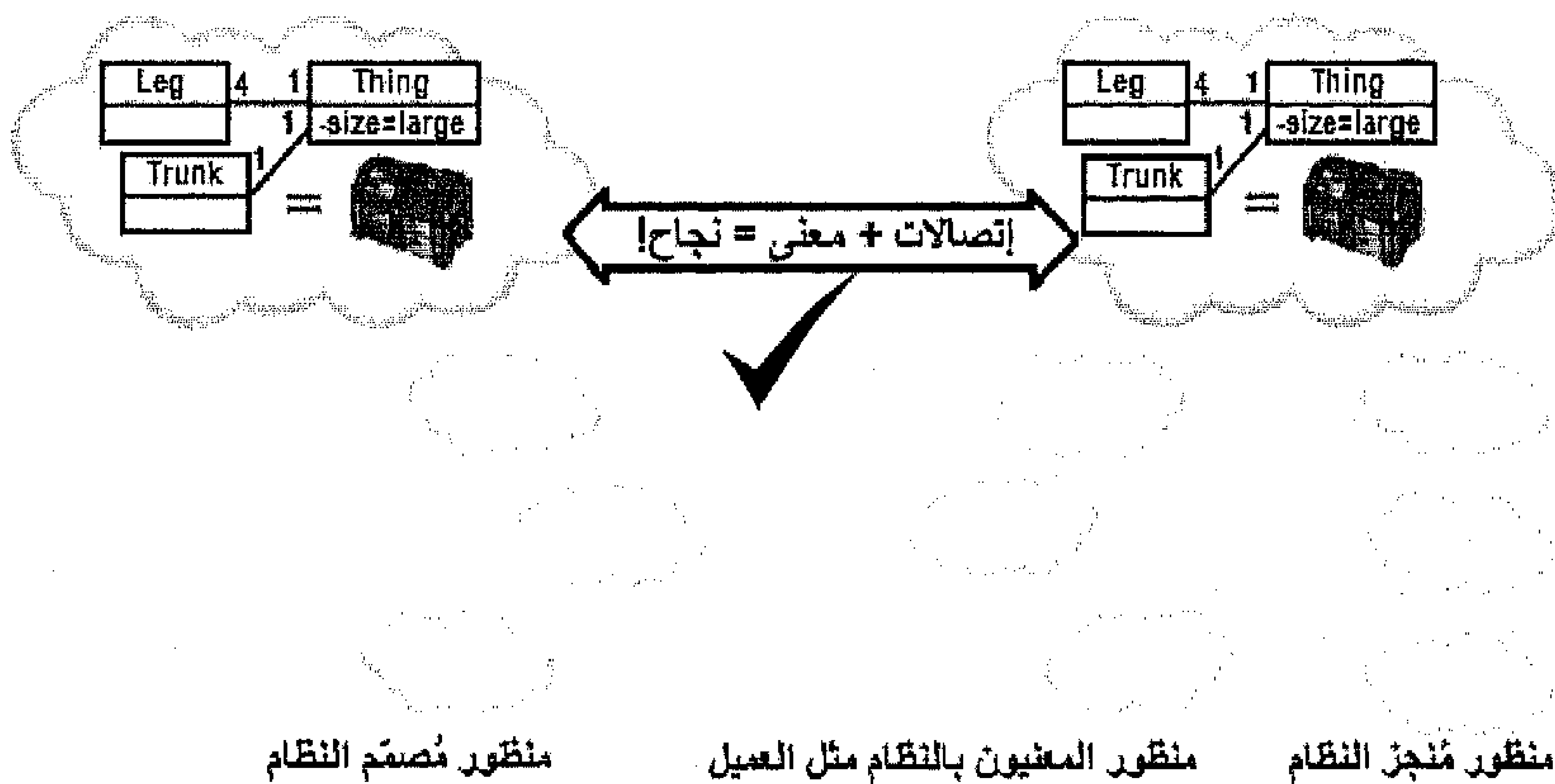
إذا كنت لا تفهم إلى الآن الترميز المستعمل في الشكل رقم (١-٥)، ربما يمكنك البدء بإدراك أن هناك بعض التفاصيل الموجودة في شفرة المثال رقم (١-١) لم تتم نمذجتها هنا. وعلى سبيل المثال، فقد تم استبعاد الإنجاز الدقيق للطريقة (play) للسماح لك بتخيل هيكل الشفرة دون تشويش الأفكار.

إن الأمر المميز لنمذجة النظام باستعمال UML، هو أن الترميز في الشكل رقم (١-٥) محدّد و معرّف المعنى. وإذا كان عليك عرض هذا المخطط على أي شخص آخر يعمل على نظامك- شرط أنه يعرف لغة UML-

فسيكون التصميم مفهوماً بشكل واضح. وهذا يمثل ميزة استعمال اللغات الرسمية للنمذجة كما هو معروض في الشكل رقم (١-٦).



شكل رقم (١-٥) يعبر عن الهيكل الساكن في ترميز رسمي للغة UML لهيكل الصنف Guitarist.



شكل رقم (١-٦) مع أية لغة نمذجة ذات معنى معرّف رسمياً، يمكن ضمان قراءة كل شخص للوصف بنفس الطريقة.

١-٢ لماذا لغة النمذجة الموحدة ٢،٠ ؟

لقد سمحت النسخة الأولى من لغة UML للناس بتناقل التصاميم بشكل واضح، وبإيصال جوهر التصميم، و حتى بأسر المتطلبات الوظيفية ومقابلتها مع حلولها البرمجية. على أية حال، لقد تغير العالم بشكل أساسي مع الاعتراف بأن نمذجة الأنظمة، أو بالأحرى نمذجة البرامج، يمكن أن تستفيد أيضاً من لغة موحدة، مثل UML.

إن العوامل الدافعة لتطوير البرامج الموجهة للمكونات component-oriented، والبنى المعمارية المنقادة بالنموذج model-driven، ولغة UML قابلة التنفيذ، والحاجة إلى مشاركة النماذج بين أدوات مختلفة، وقد وضعت متطلبات على UML لم تكن مصممة أصلاً لأخذها بالاعتبار.

بالإضافة إلى ذلك، كان الإصدار UML 1.x و كل تنقيحاته السابقة قد صمم كلغة موحدة للبشر. وعندما أصبحت مشاركة النماذج مهمة بين الآلات - خاصة بين أدوات هندسة الأنظمة بمساعدة الحاسب Computer Aided Systems Engineering (CASE) - تم مجدداً اكتشاف دوام الحاجة إلى UML 1.x. ولم تكن قواعد ترميز UML 1.x الأساسية ونموذج النموذج (meta-model) الخاص بها معرّفة رسمياً بما فيه الكفاية للتمكن من مشاركة النماذج بين الآلات.

البنى المعمارية المنقادة بالنماذج و لغة النمذجة الموحدة القابلة للتنفيذ
Model-Driven Architecture (MDA) and Executable UML

لقد شجعت نظريتان جديدتان نسبياً في تطوير النظام بإدخال عديد من التحسينات على **UML 2.0**. بإيجاز، وتوفر البنى المعمارية المنقادة بالنموذج (**MDAs**) إطار عمل يدعم تطوير نماذج مستقلة عن المنصة **Platform Independent Models (PIMs)** (أي نماذج تعبر عن النظام بطريقة معمّمة بعيدة عن اهتمامات معينة مثل لغة الإنجاز و منصة العمل).

ويمكن -بالتالي- تحويل النماذج المستقلة عن المنصة (PIMs) إلى عدة نماذج متعلقة بالمنصة **Platform Specific Models (PSMs)** تحتوي على مواصفات ملموسة لعملية نشر نظام خاص (تحتوي على تفاصيل، مثل لغة الإنجاز و بروتوكولات الاتصالات، إلخ). وتتطلب البنى المعمارية المنقادة بالنموذج (MDA) أن يكون نموذج النموذج مهيكلاً بشكل رسمي و قادراً على العمل بين أنظمة مختلفة، وذلك لإنجاز التحويلات الخاصة بها. وتوفر الآن لغة النمذجة الموحدة ٢,٠ هذا المستوى من نموذج النموذج.

للعديد من تلك الأسباب، فإن لغة النمذجة الموحدة قابلة التنفيذ تمثل وسيلة تمكن نموذج متعلق بالمنصة PSM من احتواء مقدار كاف من المعلومات الكاملة ليتمكن من إتمام تنفيذه بفعالية. وسيصبح بالإمكان في يوم ما تخيل أنك تسحب بضعة عناصر وتقوم بتكملتها لتحصل بسرعة على برنامج قابل للتنفيذ! ويتطلب محرّك لغة النمذجة الموحدة القابل للتنفيذ، أن يكون نموذجها معرّفاً بشكل جيد وكاف ليقدر على توليد النظام المنمذج وتنفيذه.

لسوء الحظ، بالرغم من أنه يفترض على **UML 2.0** توفير الآليات لجعل البنى المعمارية المنقادة بالنموذج MDA ولغة النمذجة الموحدة قابلة التنفيذ وأمور حقيقية، إلا أنه لم يتم تطوير الأدوات الداعمة لها بشكل كامل.

وبالرغم من أن UML 1.5 توصف النظام بشكل مقبول، فالنموذج الذي يصف النموذج (أي نموذج النموذج (meta-model)) قد أصبح معقداً جداً. ومثل أي نظام له تصميم معقد، وهش وصعب التوسيع، أصبحت UML معقدة وهشة وصعبة التوسيع؛ وقد حان الوقت لإعادة هيكلتها.

إن مصممي UML 2.0 كانوا حذرين جداً لضمان ألا تكون UML 2.0 غريبة عما استعمله الناس سابقاً في UML 1.x. لقد تم الاحتفاظ بعديد من المخططات الأصلية والترميزات المرتبطة بها، وقد تم توسعتها في UML 2.0، كما هو معروض في الجدول رقم (١-١). وعلى أية حال، تم إضافة أنواع جديدة من المخططات لتوسعة اللغة بشكل كاف تماماً، لتصبح باستطاعتها دعم الممارسات الأخيرة.

ومع النسخة 2.0، تم تطوير UML لتدعم التحديات الجديدة التي يواجهها اليوم صناع و مصمموا البرامج والأنظمة. فالذي بدأ قبل عدة سنوات كتوحيد للطرق المختلفة في تصميم البرامج، قد نما الآن ليصبح لغة نمذجة موحدة جاهزة، ومناسبة لتستمر في كونها اللغة القياسية لعدد كبير من المهام المختلفة و المتعلقة في تصميم البرامج والأنظمة.

جدول رقم (١-١) لوصف تنسيق تصميم النظم الواسعة، وقامت UML 2.0 بإعادة تسمية و توضيح مخططاتها لأجل التحديات الجديدة التي تواجه منمذجي النظم حالياً.

نوع المخطط	ماذا يمكن أن ينمذج؟	متى أُدرج؟	مكانه في الكتاب
حالة الاستخدام Use Case	التفاعلات بين النظام و مستخدميه أو الأنظمة الخارجية الأخرى. تفيد أيضاً في تخطيط متطلبات الأنظمة.	UML 1.x	الفصل ٢
النشاط Activity	النشاطات المتسلسلة و المتوازية في النظام.	UML 1.x	الفصل ٣
الصنف Class	الأصناف،والأنواع، والواجهات، و العلاقات التي بينهم.	UML 1.x	الفصلين ٤ و ٥
الكائن Object	الكائنات التي هي مثيلات للأصناف المعرفة في مخطط الأصناف في الترتيبات **configurations** المهمة للنظام.	بشكل غير رسمي UML 1.x	الفصل ٦
التتابع Sequence	التفاعلات بين الكائنات عندما يكون ترتيب التفاعلات مهما.	UML 1.x	الفصل ٧
الاتصال Communication	التفاعلات بين الكائنات، و الارتباطات الضرورية لدعم تلك التفاعلات.	سُمّيَ مخطط التعاون منذ UML 1.x	الفصل ٨
التوقيت Timing	التفاعلات بين الكائنات عندما يكون التوقيت مهما.	UML 2.0	الفصل ٩
ملخص التفاعل Interaction Overview	يُستعمل لجمع مخططات التسلسل، و الاتصال، و التوقيت معاً لأسر التفاعلات المهمة التي تحدث داخل النظام.	UML 2.0	الفصل ١٠
هيكل مركّب Composite Structure	داخل الصنف أو المكوّن، و يمكن أن يصف علاقات الصنف ضمن سياق محدد.	UML 2.0	الفصل ١١

نوع المخطط	ماذا يمكن أن ينمذج؟	متى أُدرج؟	مكانه في الكتاب
المكوّنات Component	المكوّنات المهمة داخل النظام و الواجهات التي تستعملها لتتفاعل فيما بينها.	في UML 1.x، وله معنى جديداً في UML 2.0	الفصل ١٢
الحُزَم Packages	التنظيم الهرمي لمجموعات الأصناف و المكوّنات.	UML 2.0	الفصل ١٣
حالة الآلة أو الحالة والانتقال State Machine	الحالة التي يأخذها كائن في جميع مراحل عمره و الأحداث التي يمكن أن تغيّر تلك الحالة.	UML 1.x	الفصل ١٤
النشر Deployment	كيفية نشر النظام أخيراً في وضعية محددة من العالم الواقعي.	UML 1.x	الفصل ١٥

١-٣ النماذج والمخططات
Models and Diagrams

يركز عديد من المبتدئين في تعلم UML على الأنواع المختلفة للمخططات المستعملة في نمذجة الأنظمة. ومن السهل جداً الافتراض أن النموذج هو: مجموعة المخططات التي تم إنشاؤها. وهذا خطأ شائع نقع فيه لأنه عندما نستعمل UML، نتفاعل عادة مع أداة للغة UML ومجموعة محددة من المخططات. لكن لا تتمحور النمذجة مع UML حول المخططات فقط؛ بل تتمحور حول أسر وتمثيل النظام كنموذج، بينما تكون المخططات في الواقع مجرد نوافذ في ذلك النموذج.

ويقوم أي مخطط بعرض بعض أجزاء النموذج وليس بالضرورة كل الأمور عنه. ويصبح هذا ذات معنى عندما لا نريد استعمال مخطط واحد لعرض كل الأمور عن النموذج دفعة واحدة، بل نريد امتلاك القدرة على تقسيم محتويات النموذج على عدة مخططات. وعلى أية حال، لا يحتاج كل أمر في النظام إلى إدراجه في مخطط خاص به ليشكل جزءاً من النموذج.

ماذا يعني ذلك؟ الأمر الأول الذي يجب فهمه هو أن النموذج عبارة عن مجموعة عناصر تتمحور خلف أداة النمذجة المستعملة والمخططات. ويمكن لكل عنصر من تلك العناصر أن يكون حالة استخدام، أو صنفاً، أو نشاطاً، أو أي مفهوم آخر تدعمه لغة النمذجة الموحدة. ويتألف النموذج من مجموعة تلك العناصر الواصفة للنظام، بما فيها الروابط التي بينها.

وإذا كان كل ما يمكن عمله هو إنشاء نموذج يتألف من عناصر، فليست عندنا أمور كثيرة لأخذها بالاعتبار. وبدلاً من أن تكون النموذج الفعلي، فتستعمل المخططات بكل بساطة كأساس يمكننا من إنشاء عناصر جديدة، يتم إضافتها لاحقاً إلى النموذج وتقوم بتنظيم العناصر ذات العلاقة في مجموعة من المنظورات (views) عن النموذج المعني.

لذا، عند استعمال أداة UML للعمل على مجموعة مخططات ضمن ترميز UML، من المفيد تذكّر أن ما تقوم بمعالجته هو رؤية محددة عن محتويات النموذج. ويمكن تغيير عناصر من النموذج داخل المخطط، لكن المخطط نفسه هو ليس النموذج - إنما هو مجرد وسيلة مفيدة لعرض بعض الأجزاء الصغيرة من المعلومات التي يحتويها النموذج.

١-٤ "درجات" استعمال لغة النمذجة الموحدة
"Degrees" of UML

يمكن استعمال UML بالقدر الكبير أو الصغير الذي تحب. وقد قام مارتن فاولر بوصف ثلاث طرق شائعة يميل إليها الناس لاستعمال UML:

استعمال UML كتصميم أولي (sketch)

يمكـن اسـتعمال UML لإنشـاء تصـاميم أوليـة مختصـرة لتوصـيل النقاط الرئيسة. وتستعمل تلك التصـاميم مـرة واحـدة ثـم يتم رميهـا ، حيـث يمكـن أن ترسم على لوحة بيضاء ثم تمسح بعد ذلك.

استعمال UML كطابعة كربونية (blueprint)

توفر UML توصيفاً مفصّلاً للنظـام باسـتعمال مخططاتهـا. ولـن يـتم التخلص من هذه المخططات بعد استعمالها ولكن ستيم توليدها باستعمال أداة للغـة UML. بشـكل عـام، تـرتبط هـذه المنهجيـة بالأنظمـة البرمجيـة ، وعـادة مـا تقتضـي اسـتعمال الهندسـة الأماميـة والعكسـية لإبقـاء النمـوذج متزامناً مع شفرة البرامج.

استعمال UML كلغة برمجة

وذلك بالتحويل المباشر لنموذج UML إلى شفرة قابلة للتنفيـذ (لـيس أجزاء من الشفرة فقط كما هو الحال مع الهندسة الأمامية)؛ مما يعني أنـه قد تم نمذجة كل سمة في النظام. ويمكـن نظرياً إبقاء النموذج غير معـرّف واسـتعمال التحـويلات، وتوليـد الشـفرة لنشـره واسـتخـدامه في بيئـات مختلفة.

وتعتمـد المنهجيـة المستعملة علـى نـوع التطبيـق الـذي تبنيـه؛ بـأي دقّـة ستتم مراجعـة التصـميم؟ إذا كنـت تطـور نظامـاً برمجيـاً ، وعمليـة تطـوير البرامج التي تستعمل في حال كونه برنامجاً عادياً.

وفي بعض المجالات الصناعية، مثل الصناعات الطبيـة و الدفاعيـة ، تشهد مشاريع البرامج ميلاً نحو استخدام UML كطابعـة كربونيـة ، لأن

هذه المجالات تتطلب مستوى عالٍ من الجودة. وتتم مراجعة تصميم البرنامج بكل تأنٍ بسبب خطورة هذه المهمة: فمثلاً، لا تريد لآلة مراقبة قلبك أن تعرض فجأة "شاشة الموت الزرقاء".

ويمكن أن تطلق بعض المشاريع مع قليل من النمذجة. وفي الحقيقة، تجد بعض الصناعات التجارية أن الإفراط في النمذجة شيء متعب و يقلص الإنتاجية. ومع هذا النوع من المشاريع، ومن الأفضل استعمال UML كتصميم أولي، حيث يكون النموذج محتوياً على بعض المخططات المعمارية وقليل من الأصناف، ومخططات التتابع لتمثيل النقاط الرئيسية.

١-٥ لغة النمذجة الموحدة و عملية تطوير البرامج
UML and the Software Development Process

عند استعمال UML لنمذجة نظام برمجي، فتتأثر "درجة استعمال UML" جزئياً بعملية تطوير البرامج المستعملة.

إن عملية تطوير البرامج هي طريقة مستعملة لبناء البرامج؛ حيث يتم تحديد قدرة عملية التطوير، وكيفية بنائها، ومَن يعمل على ماذا، والأطر الزمنية لكل النشاطات. وتهدف تلك العمليات إلى إدخال الانضباط والتوقعية في عملية تطوير البرامج، وذلك من أجل زيادة فرص نجاح مشاريع التطوير. وبما أننا سنستعمل لغة UML لنمذجة البرامج، فستكون جزءاً مهماً من عملية تطوير البرامج.

وتتضمن عمليات تطوير البرامج المشهورة جداً الطرق التالية:

طريقة الشلال Waterfall

تحاول طريقة الشلال تثبيت المتطلبات في بداية دورة حياة المشروع. وبعد تجميع المتطلبات، يتم إنجاز تصميم البرنامج بالكامل. وعندما يصبح

التصميم كاملاً، تتم كتابة البرامج. ومشكلة هذه الطريقة هي تأثّرها الكارثي بأي تغيير قد يحدث في المتطلبات.

الطريقة التكرارية Iterative

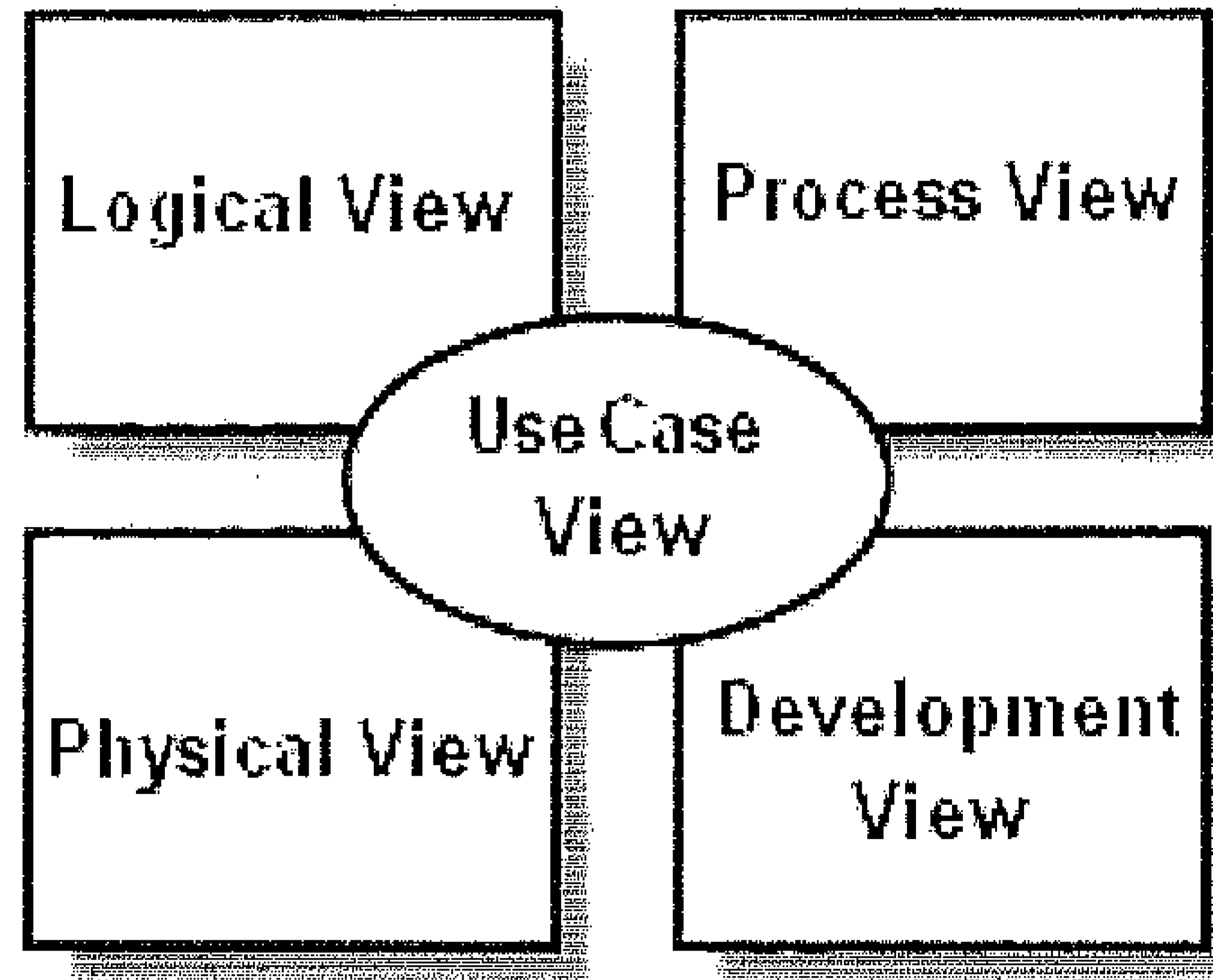
تحاول الطرق التكرارية مواجهة عيوب طريقة الشلال بقبول إمكانية حدوث التغيير بالمتطلبات و احتضانه. إن العملية الموحدة Unified Process هي عملية تكرارية معروفة بشكل جيد، وهي متعددة المراحل، حيث تحتوي كل مرحلة على عدد من النشاطات التالية: المتطلبات، والتصميم، والإنجاز (كتابة شفرة المصدر). وتشمل الطرق التكرارية طيفاً واسعاً من المنهجيات، (مثل العمليات التكرارية الذكية agile iterative processes)، ويمكن أن تمتد هذه العمليات من استعمال لغة UML كتصميم أولي إلى استعمالها كطابعة كربونية.

الطرق الذكية Agile Methods

تستعمل الطرق الذكية التكرارات في وضعيات فجائية قصيرة جداً، حيث تحاول تقليل الخطر من خلال الامتلاك الدائم لنظام شغال متنامي القدرات. ولقد قدمت المنهجيات المنطوية تحت هذا التصنيف بعضاً من ممارسات التطوير الأكثر أهمية، مثل البرمجة المزدوجة pair programming (تقنية تطوير برامج حيث يَعْمَلُ برنامجان معاً على لوحة مفاتيح واحدة)، والتطوير المُنقاد بالاختبار test-driven development. وتشدد الطرق الذكية على استعمال UML كتصميم أولي.

١-٦ منظورات نموذجك
Views of Your Model

يوجد عدد من الوسائل لتقسيم مخططات UML الخاصة بنموذجك إلى تصوّرات أو منظورات تأسر جوانب محددة من نظامك. وفي هذا الكتاب، نستعمل نموذج المنظورات ٤+١ الخاص بكرتشن Kruchten لمساعدتك في عرض الدور الذي يؤديه كل نوع من المخططات في النموذج العام، وكما هو معروض في الشكل رقم (١-٧).



شكل رقم (١- ٧) نموذج المنظورات E+١ الخاص بفليب كيرتشن **Philippe Kruchten**.

يقسم نموذج المنظورات ٤+١ أي نموذج إلى مجموعة منظورات، يأسر كل واحد منها سمة محددة في نظامك وهي:

المنظور المنطقي Logical View

يصف التوصيفات المجردة لأجزاء النظام. ويُستعمل عادة لنمذجة الأجزاء المكونة للنظام ولتبيان كيفية تفاعلها فيما بينها. وتتضمن مخططات UML المؤلفة لهذا المنظور مخططات الأصناف classes، والكائنات objects، وحالة الآلة state machine، والتفاعل interaction.

المنظور العملياتي Process View

يصف العمليات التي داخل نظامك. وهو مفيد جداً عند إظهار ما يجب حدوثه داخل نظامك. وعادة ما يضم هذا المنظور مخططات النشاط activity.

المنظور التطويري Development View

يصف كيف تكون أجزاء نظامك منظمة في وحدات و مكوّنات. وهو مفيد جداً لإدارة الطبقات داخل معمارية نظامك. وعادة ما يضم هذا المنظور مخططات الحُزَم packages، ومخططات المكوّنات components.

المنظور المادي Physical View

يصف كيفية تطبيق تصميم النظام، كما هو موصوف في المنظورات الثلاثة السابقة، في الواقع على شكل مجموعة كيانات من العالم الحقيقي. تعرض المخططات التي في هذا المنظور كيف تتدرج الأجزاء المجردة في النظام النهائي المنشور. وعادة ما يضم هذا المنظور مخططات الانتشار deployment.

منظور حالة الاستخدام Use case View

يصف تشغيل النظام كونه مُنَمْذَجاً من خلال تصور العالم الخارجي. إن هذا المنظور ضروري لوصف ما يجب أن يعمله النظام. وتعتمد كل المنظورات الأخرى على منظور حالة الاستخدام من أجل توجيههم، ولهذا السبب تمت تسمية النموذج ٤+١. وعادة ما يضم هذا المنظور مخططات حالة الاستخدام، ومخططات التوصيفات descriptions، والملخصات overviews.

يوفر كل منظور تصوراً مختلفاً ومهماً عن نموذجك. وإذا كنت تتساءل لماذا عليَّ الاهتمام بهذا؟ وكأنك تقرأ عن ترميز أو مخطط محدد، فارجع إلى المنظور الذي يوفره هذا الترميز، أو المخطط لفهم سبب الحاجة إليه.



لمزيد من المعلومات حول نموذج المنظورات ٤+١ الخاص بكرتشن، تحقق من البحث المقدم منه: "Architectural Blueprints The '4+1' View Model of Software Architecture"، على الموقع
http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf
وللحصول على استعراض عام عن هذا النموذج يمكنك زيارة الموقع
http://www-128.ibm.com/developerworks/wireless/library/wi-arch11/.

١-٧ أول تذوّق من لغة النمذجة الموحدة
A First Taste of UML

قبل الوثب داخل الأنواع المختلفة للمخططات المؤلفة للغة UML، تحتاج إلى معرفة عنصرين من ترميزات UML يتم استعمالهما في مجمل النموذج: الملاحظات notes، والحاشيات stereotypes.

١-٧-١ الملاحظات notes

تسمح لك الملاحظات بإدخال تعليقات إضافية لم يتم أسرها في مخططاتك. ويمكنك كتابة أي شيء تريده داخل الملاحظة من أجل توضيح مخططاتك، وهي تشبه تعليقات شفرة المصدر. وترسم الملاحظات باستعمال ترميز المستطيل المثني الزاوية العليا عن اليمين كما هو معروض في الشكل رقم (١-٨).

A Note

شكل رقم (١-٨) ملاحظة في ترميز لغة النمذجة الموحدة.

يمكن وضع الملاحظات على المخطط بشكل منفصل أو متصل بجزء محدد منه، كما هو معروض في الشكل رقم (١-٩).

تقوم هذه الملاحظة بمجرد عرض كيفية ربطها بأجزاء من محتوى المخطط، في هذه الحالة الصنف BlogAccount

BlogAccount

شكل رقم (١- ٩) ملاحظة متصلة بعنصر آخر من المخطط باستعمال خط منقط.

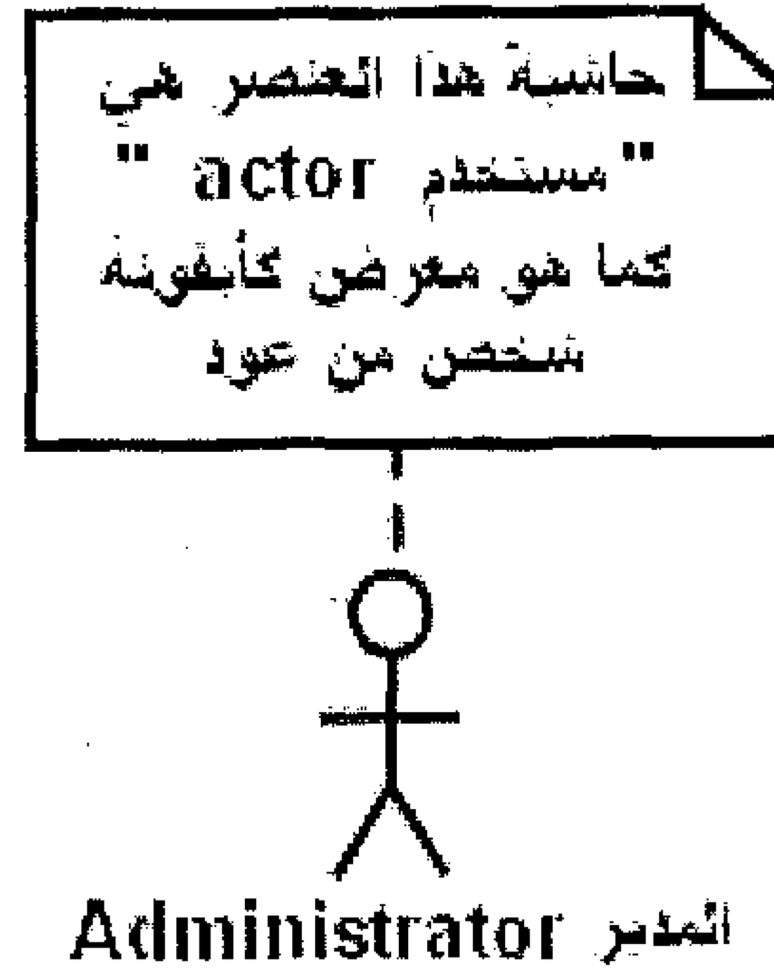
في هذا الكتاب، تستعمل الملاحظات لتوضيح أمرٍ ما بخصوص المخطط. وتشكل الملاحظات مجرد مساعدات يوفرها مصمم المخطط لقارئه؛ وهي لا تغيّر في معنى المخطط أو النموذج الأساسي على الإطلاق.

١-٧-٢ الحاشيات Stereotypes

تشير الحاشيات إلى استعمال خاص، أو هدف محدد حيث يمكن تطبيقها على معظم عناصر ترميزات UML. وتقوم الحاشيات بتغيير معنى العنصر وبوصف دوره داخل نموذجك.

ويمكن- أحياناً - أن ترتبط الحاشية بأيقونة محددة، مثل أيقونة رمز المستخدم actor، كما في الشكل (١-١٠)، المكوّنة من

خطوط ترسم هيئة شخص (لمعرفة المزيد عن المستخدمين، انظر الفصل الثاني).



شكل رقم (١- ١٠) يظهر المدير Administrator في دور مستخدم لأنه يستعمل ترميز شكل الشخص المرتبط بتلك الحاشية.

ويمكن أن لا تتواجد دائماً أيقونة خاصة بالحاشية، وأحياناً تحتل الحاشيات حيزاً كبيراً جداً؛ حيث تنثر الفوضى في المخطط. وفي هذه الحالات، يتم عرض الحاشية داخل علامتي الحصر « و » مع وضع اسم الحاشية بينهما، كما في الصيغة «**stereotype_name**» و كما هو معروض في الشكل رقم (١-١١). وعلى أية حال، وبما أن علامتي الحصر تتطلبان مجموعة رموز موسعة، فيمكن استبدالها بأقواس الزاوية على النحو **<<stereotype_name>>**.



شكل رقم (١-١١) يمثل العنصر مدير مستخدماً actor، ولكن تم تحديد حاشيته هنا باستخدام اسم مع علامات الحصر بدلاً من استخدام أيقونة.

لا يوجد حدّ معين لعدد الحاشيات التي يتم ربطها بالعنصر؛ ويفضل أحياناً تحديد أكثر من حاشية واحدة كما هو مبين في الشكل رقم (١-١٢).

<<actor, person>>
Administrator

شكل رقم (١-١٢)تم هنا تحديد حاشية للمدير على أنه مستخدمactor وشخصperson معاً.

تقوم مواصفات UML بتعريف مجموعة حاشيات "قياسية" أو معرّفة مسبقاً. وتتضمن بعض الحاشيات القياسية الأكثر إفادة:

١-٧-٢-١ حاشية تطبق على الأصناف (انظر إلى الفصلين الرابع والخامس)

- فائدة (**utility**): تمثل صنفاً يزود مستخدميه بخدمات مفيدة من خلال الطرق الساكنة، مثل الصنف Math بلغة البرمجة جافا.

١-٧-٢-٢ الحاشيات المطبقة على المكوّنات (انظر إلى الفصل الثاني عشر)

- خدمة (**service**): هي مكوّن وظيفي يقوم بحساب قيمة معينة و ليس له حالة معروفة؛ ويمكن استعماله لتمثيل خدمة وِبْ.
- نظام فرعي (**subsystem**): هو مكوّن ضخم يكون فعلياً عبارة عن نظام تابع أو نظام فرعي لنظام أكبر.

١-٧-٢-٣ الحاشيات المطبقة على الأدوات المصتعة artifacts (انظر الفصل الخامس عشر)

- قابل للتنفيذ (**executable**): عبارة عن ملف قابل للتنفيذ، مثل الملفات ذات الامتداد **exe**.

- ملف (**file**): عبارة عن ملف مستعمل من قبل نظامك؛ يمكن أن يكون ملف ترتيب أو تهيئة configuration، مثل الملفات ذات الامتداد txt.
- مكتبة برمجية (**library**): عبارة عن ملف مكتبة ساكنة أو ديناميكية؛ ويمكنك استعماله لنمذجة ملفات المكتبات البرمجية ذات الامتدادات dll أو jar.
- مصدر (**source**): عبارة عن ملف مصدري يحتوي على برنامج شفرة المصدر، مثل الملفات البرمجية ذات الامتدادات java أو cpp.

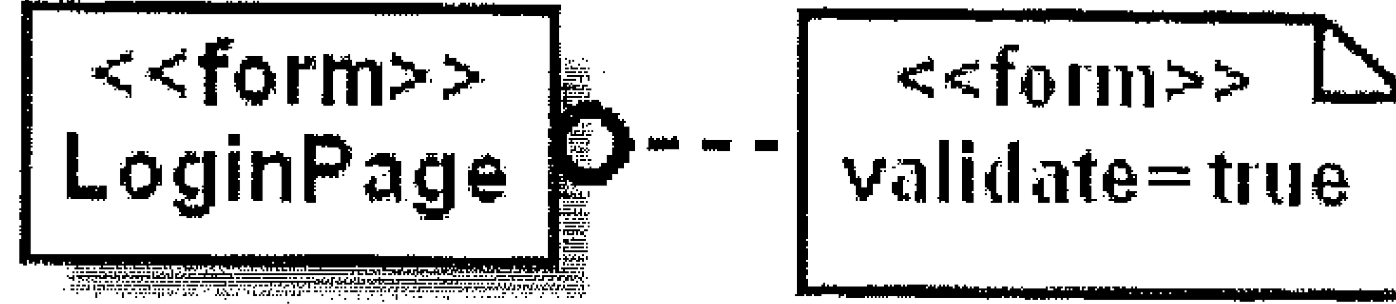
١-٧-٢-٤ القيم الملحقة Tagged values

يمكن أن تحتوي الحاشيات على معلومات إضافية تتعلق بالعنصر الذي تطبق عليه. ويتم تحديد هذه المعلومات الإضافية باستعمال القيم الملحقة.

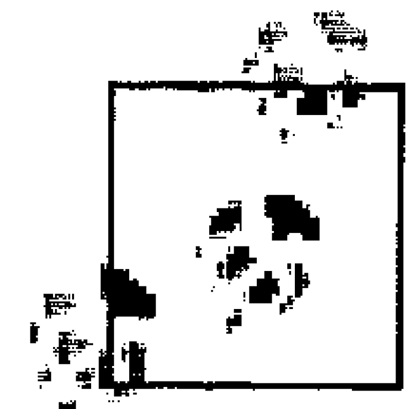
وترتبط القيم الملحقة بحاشية معينة. لنفترض أن لديك عنصراً في نموذجك يمثل صفحة الدخول LoginPage إلى موقع وِبْ، ووضع له الحاشية استمارة <<form>>، فإن الحاشية استمارة تحتاج إلى معرفة إذا كان عليها المصادقة validate على محتويات هذه الاستمارة أم لا. في هذه الحالة، يجب الإعلان عن قرار المصادقة كقيمة ملحقة بالحاشية استمارة؛ لأن القيمة الملحقة مرتبطة بالحاشية المطبقة على العنصر، وليس بالعنصر نفسه.

وترسم القيمة الملحقة على المخطط باستعمال ترميز مشابه للملاحظات، ولكن يحتوي المستطيل المثني على اسم أي حاشيات وإعدادات لأي قيم ملحقة مرتبطة بها. وبالتالي يتم ربط ملاحظة القيمة الملحقة بعنصر الحاشية باستعمال خط منقّط مع دائرة عند نهاية العنصر،

كما هو معروض في الشكل رقم (١-١٣). (لقد تم اقتباس هذا المثال من كتاب [O'Reilly] "UML 2.0 in a Nutshell").



شكل رقم (١-١٣) أضيف للحاشية form قيمة ملحقة مرتبطة بها حيث تم تسميتها validate وإعطائها القيمة صح true.



في UML 2.0 ، يتم تعريف الحاشيات و القيم الملحقة باستعمال مظاهر الأقلمة أو التكييف profiles. انظر إلى الملحق ب لمعرفة المزيد عن الحاشيات و كيفية إنشاء أدوار لعناصر نموذجك.

١-٨ هل ترغب بمعلومات إضافية؟
Want More Information?

تتمحور المرحلة التالية في الانتقال إلى الفصل الثاني والبدء بتعلم UML. إذا كنت متحمساً لمعرفة لمحة تاريخية عن UML ، ويمكنك الذهاب إلى الملخص التاريخي عن UML في الملحق ج.

إن UML هي لغة موجزة ولكنها تشكل موضوعاً كبيراً. وخلال تعلمك UML ، يمكنك الاستعانة بقراءة البحوث والوثائق المتوفرة في الموقع http://www.omg.org الخاص بالمؤسسة Object Management Group (OMG).

نمذجة المتطلبات:
حالات الاستخدام

MODELING REQUIREMENTS: USE CASES

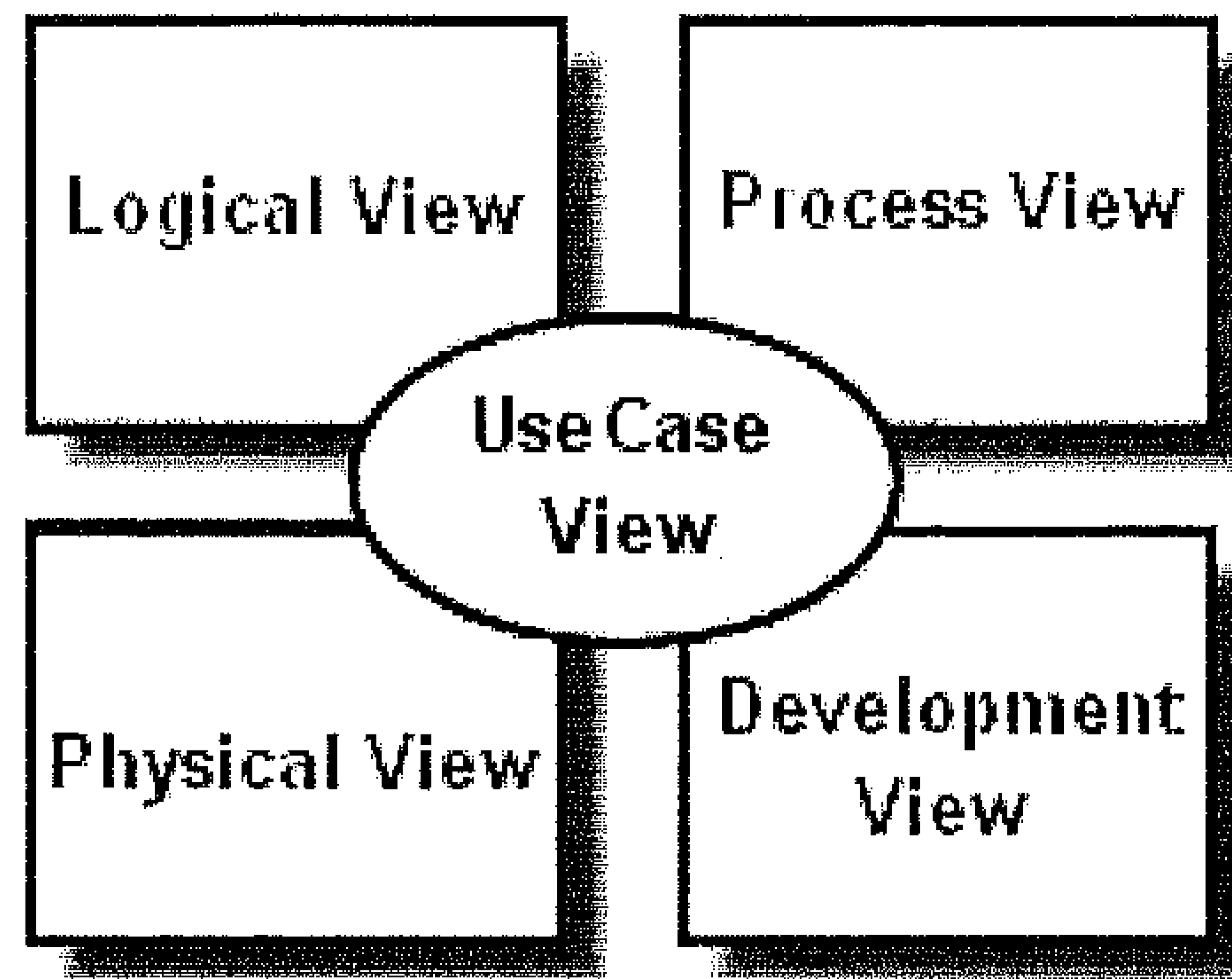
تخيّل في صباح يوم بداية الأسبوع، وأنه يومك الأول في العمل على مشروع جديد، ودخل فجأة الأشخاص العاملون على المتطلبات لأخذ القهوة الصباحية، وتركوا لك ٢٠٠ صفحة من الوثائق عن المتطلبات التي عملوا عليها خلال الأشهر الست الماضية. وكانت أوامر رئيسك بسيطة: "انهض بفريقك للإسراع بالعمل على هذه المتطلبات كي يصبح باستطاعة الكل البدء بتصميم النظام". يا لها من بداية أسبوع سعيدة!

لنُصعب الأمور قليلاً، افترض أن المتطلبات ما زالت غير واضحة نسبياً، و كانت مكتوبة بلغة المستخدم - اللغة الطبيعية المربكة والغامضة - بدلاً من لغة يمكن أن يفهمها الأشخاص الآخرون المعنيون بالنظام بسهولة. (انظر في الفصل الأول إلى القسم "الإسهاب، الغموض، التشويش: النمذجة مع اللغات غير الرسمية"، حيث المزيد من المعلومات حول موضوع مشاكل النمذجة باللغات الطبيعية واللغات غير الرسمية).

ما هي الخطوة القادمة؟ كيف ستأخذ هذه المجموعة الضخمة من المتطلبات المعرفة بتراخ وتنقيتها أو تنقيحها لتصبح ضمن بنية تخص مصمميك، من دون فقدان التفاصيل المهمة؟

كما رأيت في الفصل الأول، فلغة UML هي الجواب عن هذين السؤالين معاً. وعلى وجه التحديد، تحتاج إلى العمل مع الأشخاص الآخرين العاملين على نظامك، لإنتاج مجموعة كاملة من المتطلبات و شيء جديد يسمى "حالات الاستخدام use cases".

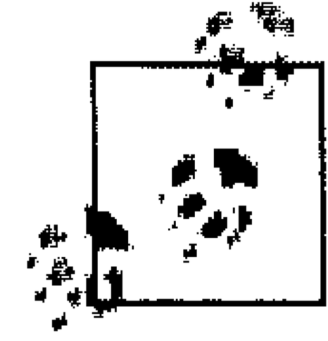
إن حالة الاستخدام هي حالة أو وضعية يتم استخدام نظامك فيها لإنجاز متطلب واحد أو أكثر من متطلبات المستخدم؛ وتأسر حالة الاستخدام جزءاً من الوظيفة التي يوفرها النظام. وتقع حالات الاستخدام في قلب نموذجك، كما هو معروض في الشكل رقم (٢-١)، لأنها توجه وتؤثر على كل العناصر الأخرى داخل تصميم نظامك.



شكل رقم (٢-١) تؤثر حالات الاستخدام على كل الجوانب الأخرى لتصميم نظامك؛ فهي تأسر ما هو متطلب، أما المنظورات الأخرى فتبين كيفية تلبية هذه المتطلبات.

تشكل حالات الاستخدام نقطة بداية ممتازة لكل جزئية من جزئيات تطوير نظام بالأسلوب الكائني التوجه، كالتصميم، والاختبار، والتوثيق. إنها تصف متطلبات النظام بدقة للناظرين من الخارج؛ فهي تحدد القيمة التي يعطيها النظام للمستخدمين. وبما أن حالات الاستخدام تمثل

المتطلبات الوظيفية لنظامك، فيجب أن تكون أول المخرجات الجديّة من نموذجك بعد البدء بمشروع معين. وعلى كل حال، كيف يمكنك البدء بتصميم نظام إذا كنت لا تعرف ما المطلوب عمله؟



تحدد حالات الاستخدام ما يفترض بالنظام أن ينفذه فقط، أي المتطلبات الوظيفية للنظام. إنها لا تحدد ما لن يقوم بتنفيذه النظام، أي المتطلبات غير الوظيفية للنظام. وغالباً ما تتضمّن المتطلبات غير الوظيفية أهداف الأداء ولغات البرمجة، ... إلخ.

عندما تعمل على متطلبات نظام معين، تظهر غالباً أسئلة، مثل هل للنظام متطلباً خاصاً؟ وتشكل حالات الاستخدام وسيلة جيدة لإظهار تلك الفجوات الكامنة في متطلبات المستخدم إلى بداية المشروع.

يشكل هذا الأمر إعانة حقيقية لمصمم النظام؛ لأن التعرّف المبكّر على أي فجوة أو نقص في فهم الأمور الخاصة بتطوير المشروع، سيكلف وقتاً ومالاً أقل بكثير من عدم اكتشاف المشكلة حتى الوصول لنهاية المشروع. وبمجرد تحديد أي فجوة، يتم الرجوع إلى الأشخاص المهتمين بالنظام - من زبائن ومستخدمين - للحصول على المعلومات الناقصة.



من الأفضل عرض المتطلبات كحالات استخدام، مما يسمح للأشخاص المعنيين من رؤية مدى أهمية هذه المتطلبات بالنسبة للنظام. وهذا يساعدهم في استبعاد حالات الاستخدام غير المهمة، وبالتالي توفير المال والوقت معاً.

ويساعد تحديد أولوية كل حالة وأهميتها استخدامها في إدارة حجم العمل على المشروع. ويمكن إسناد حالات الاستخدام إلى الفرق أو الأفراد ليتم إنجازها، وبما أن حالة الاستخدام تمثل قيمة ملموسة

للمستخدم، فيمكنك حينها تعقب تقدّم المشروع من خلال حالات الاستخدام المسلّمة. إذا تعرض مشروع معين للخلل في الجدول الزمني، ويمكن التخلص من أو تأخير حالات الاستخدام الأقل أهمية لتسليم ذوات القيمة الكبرى بأقرب وقت.

وإضافة إلى ما ذكرنا، تساعد حالات الاستخدام - أيضاً - على بناء اختبارات النظام. وتشكل حالات الاستخدام نقطة بداية ممتازة لبناء اختبار الحالات والإجراءات؛ لأنها تأسر بدقة متطلبات المستخدم ومعايير النجاح. وأي وسيلة أفضل لاختبار نظامك من استعمال حالات الاستخدام التي تأسر أصلاً ما يريده المستخدم بالدرجة الأولى؟

٢-١ أسر متطلبات النظام
Capturing a System Requirement

يكفي كلام نظري إلى الآن؛ دعنا نلقي نظرة على مثال بسيط، لنفترض أننا عرّفنا متطلبات نظام إدارة محتويات مدونة weblog Content Management System (CMS) (موقع وِبْ يهتم بتسجيل يوميات مبوبة يكتبها أشخاص مشتركين بالمدونة).

المتطلب أ-١

يجب أن يسمح نظام إدارة المحتويات للمدير بإنشاء حساب مدونة جديد، بشرط أن يتم التثبت من المعلومات الشخصية المفصلة عن مستخدم جديد للمدونة **blogger**، وذلك باستعمال قاعدة بيانات الناشر، أو الكاتب المعتمد **author** (المدوّنون).

لا يوجد في الحقيقة "أفضل وسيلة" محددة للبدء بتحليل المتطلب أ-١، لكن كخطوة أولى مفيدة، يمكنك النظر إلى الأشياء things التي

تتفاعل مع نظامك. ومع حالات الاستخدام تسمى هذه الأشياء الخارجية بالمستخدمين actors.



إن المصطلحين مؤكّدٌ shall ومأمولٌ should لهما معنى دقيق وخاصاً عندما يتعلق الأمر بالمتطلبات. إن المتطلب "مؤكد shall" يجب أن يتم إنجازه؛ إذا لم تكن الخاصية المنفذة لمتطلب "مؤكد shall" موجودة في النظام النهائي، فلا يلبي النظام هذا المتطلب. أما المتطلب "مأمول should" فيدل على أن هذا المتطلب ليس مهماً بدرجة كبيرة بالنسبة لعمل النظام، ولكنه يبقى مرغوباً به. وإذا كان تطوير النظام يسير في مشاكل قد تتسبب بتأخر التسليم، فيتم التضحية في البداية بالمتطلبات المأمولة should.

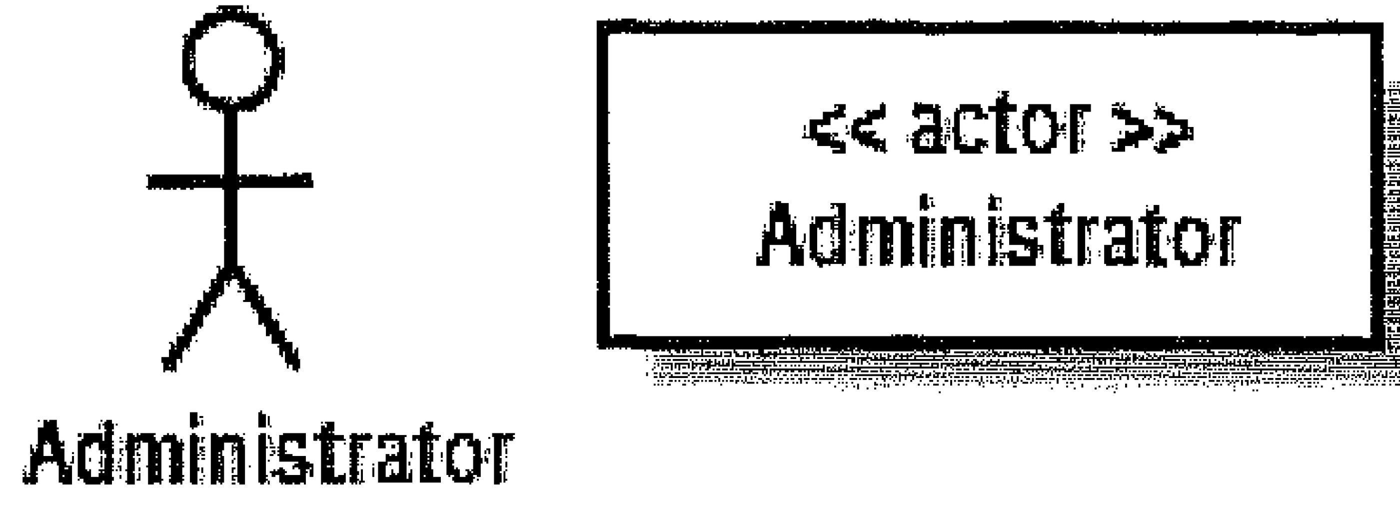
خصائص المدونة Blog Features

بدأ المصطلح weblog، والذي يشار إليه كالمصطلح blog، كصفحات وِبْ مصانة بصورة شخصية، وذلك من أجل أن يكتب فيها كتبة عن أي شيء يريدونه. وفي هذه الأيام، غالباً ما تُحزّم المدونات في نظام إدارة محتويات شامل CMS. ويقدّم مستخدمو المدونة bloggers مداخل جديدة في النظام (تسمى تدوينات blogging)، يخصّص مدراء النظام حسابات لكتابة التدوينات، وتتضمّن عادة هذه الأنظمة خصائص متقدّمة، مثل RSS feed. ويمكن لمدونة معلن عنها بشكل جيد جذب آلاف القرّاء (انظر إلى مدونات أورايلي http://weblogs.oreillynet.com).

٢-١-١ النطاق الخارجي لنظامك: المستخدمون

Outside Your System: Actors

يتم رسم المستخدم في ترميز UML باستعمال إما "شكل شخص من قضبان" أو صندوق ذو حاشية (انظر إلى "الحاشيات" في الفصل الأول) يعنون باستعمال اسم مناسب، كما هو معروض في الشكل رقم (٢-٢).



شكل رقم (٢-٢) يحتوي المتطلب أ- ١ على المستخدم مدير الذي يتفاعل مع النظام لإنشاء حساب مدونة.

يأسر الشكل رقم (٢- ٢) دور المدير Administrator كما هو موصوف في المتطلب أ-١. إن النظام المنمذج هو نظام إدارة محتوى CMS؛ ويشير وصف المتطلب إلى أن المدير يتفاعل مع النظام لإنشاء حساب مدونة جديد. ويتفاعل المدير مع النظام وهو ليس جزءاً منه؛ بالتالي، ويتم تعريف المدير على أنه مستخدم.

إن تحديد مستخدمي النظام أمر دقيق وصعب، ويتم أحياناً تعلمه بشكل أفضل من خلال التجربة. حتى تكون قد اكتسبت بعضاً من تلك التجربة، يعرض الشكل رقم (٢-٣) تقنية بسيطة لتحليل "أمراً محدداً" في متطلباتك وتحديد إن كان هذا الأمر مستخدماً أم لا.

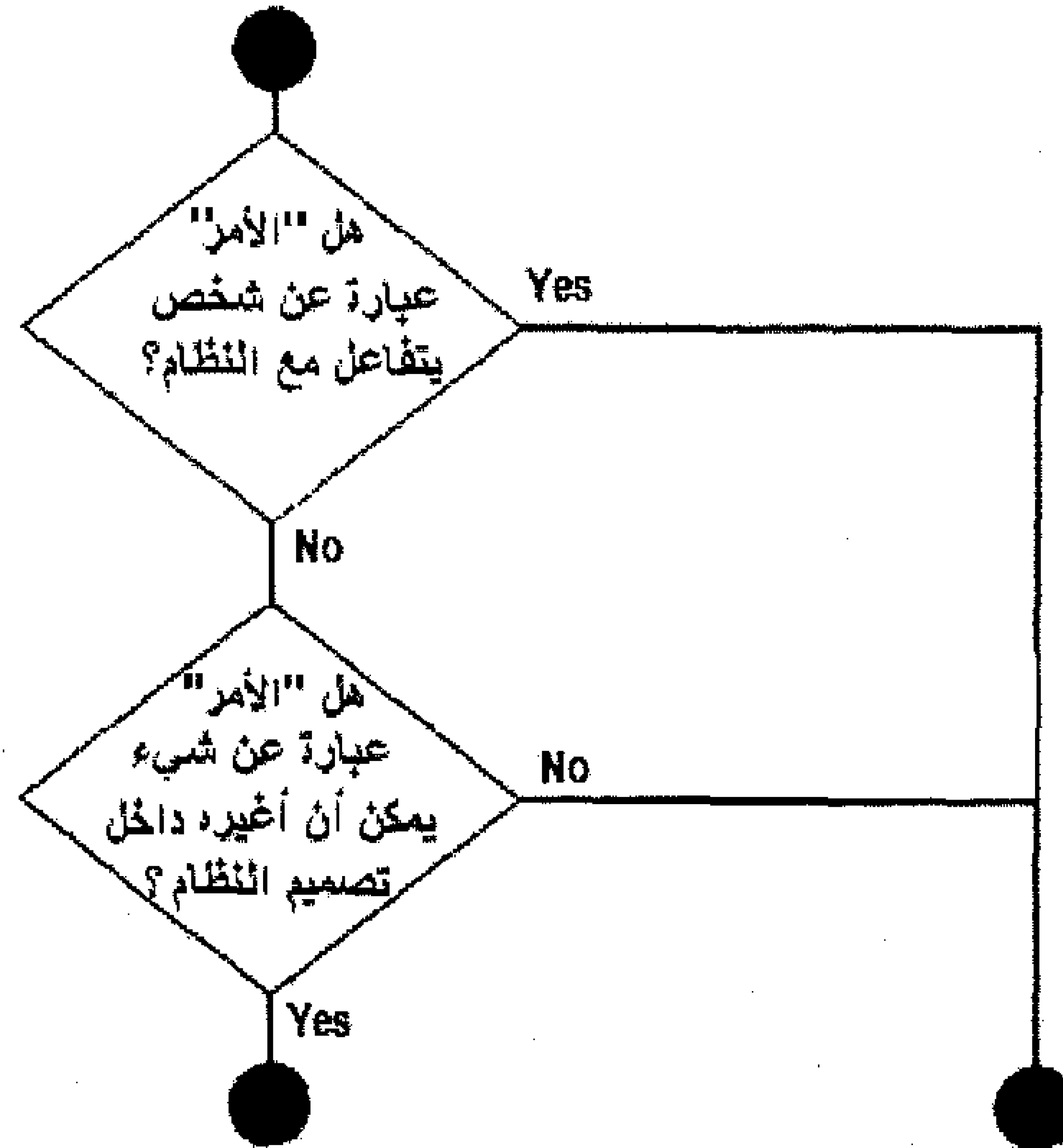
ليس على المستخدمين أن يكونوا أناساً فعليين. بالرغم من أن المستخدم ربما يكون إنساناً، فبإمكانه أيضاً أن يكون نظاماً لطرف ثالث، كما هو الحال مع تطبيق التجارة من شركة إلى شركة (B2B) Business-to-Business. تخيل المستخدم كأنه صندوق أسود: لا يمكنك تغيير المستخدم ولا يهمك كيفية عمله، لكن يجب عليه أن يتفاعل مع نظامك.

كيفية اختيار الأسماء What's in a Name؟

من الجدير بك أن تعتني كثيراً بتسمية مستخدميك. وأفضل طريقة هي باختيار اسم يفهمه عميلك ومصممو نظامك معاً. حيثما يكون ممكناً، قم باستعمال المصطلح الأصلي للمستخدم كما تم تحديده في متطلبات عميلك؛ وبهذه الطريقة، ستكون حالات استخدامك على الأقل مألوفة لدى عملائك. وتقوم هذه الطريقة أيضاً بإراحة مصممي النظام من خلال عملهم ضمن سياق فريد للنظام.

٢-١-١-١ المستخدمون المخادعون Tricky actors

ليس كل المستخدمين عبارة عن أنظمة خارجية واضحة أو أناس تتفاعل مع نظامك. ونظام الساعة هو مثال شائع لمثل هذا المستخدم المخادع. ويدل الاسم وحده على أن الساعة هي جزء من النظام، لكن هل هي حقاً كذلك؟



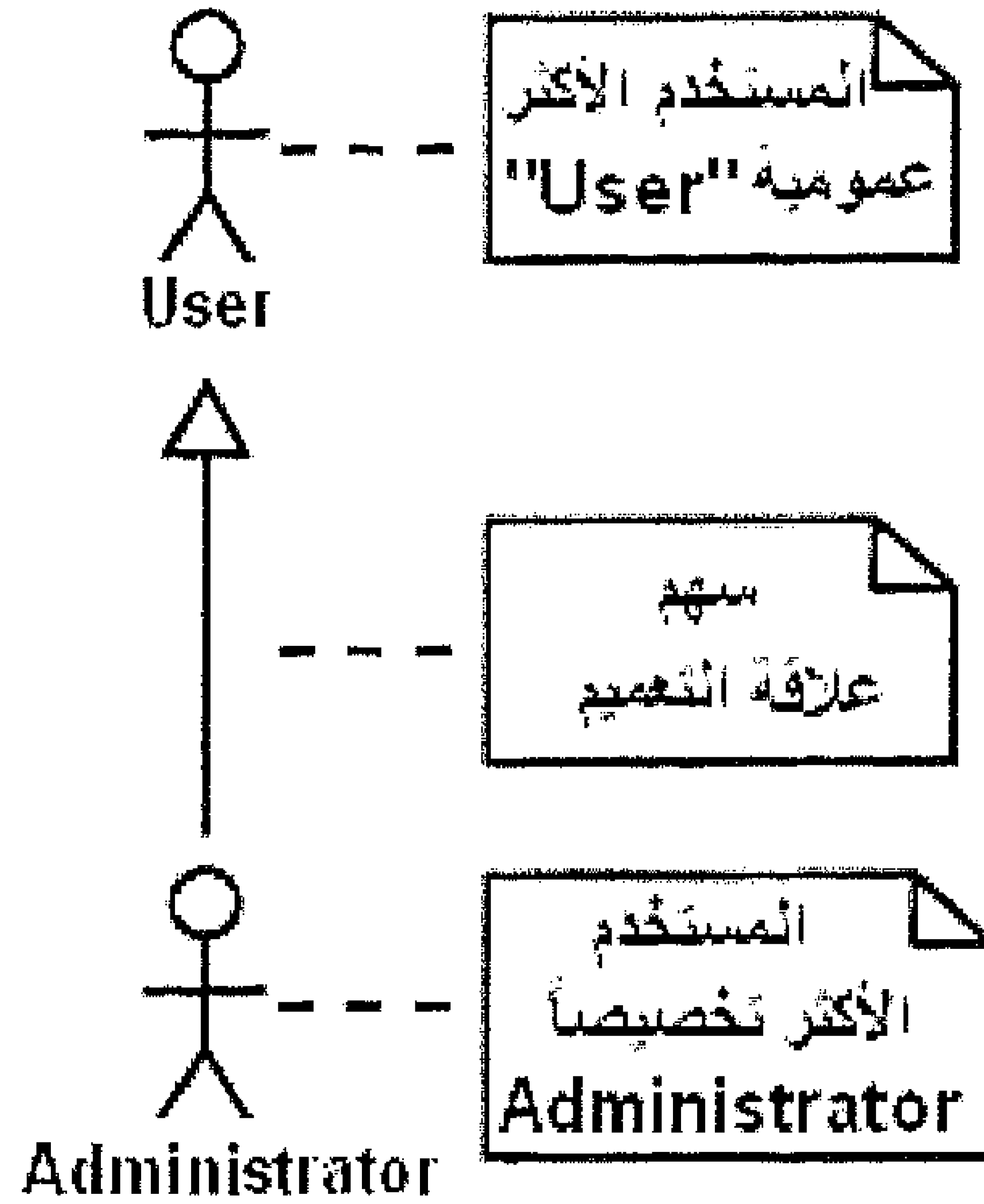
شكل رقم (٢-٣) يوجد هنا سؤالان تسألهما نفسك عندما تحاول تحديد مستخدم معين.

يدخل نظام الساعة في اللعبة عندما يقوم باستدعاء بعض السلوكيات داخل نظامك. ومن الصعب تحديد إذا كان نظام الساعة هو مستخدم؛ لأن الساعة ليست خارج نظامك بشكل واضح. والظاهر أن نظام الساعة يتم عادة وصفه جيداً على أنه مستخدم؛ لأنه أمر لا يمكنك التأثير فيه. من ناحية أخرى، إن وصف الساعة على أنها مستخدم سيساعدك عند إثبات أن نظامك يحتاج إلى تنفيذ مهمة ما ترتكز على الوقت الحالي.

من المغري أيضاً التركيز على أن مستخدمي أنظمتك هم ببساطة مستخدمو نموذجك، لكن لا تنس الأشخاص الآخرين، مثل مدقّقي الحسابات، المنصّبين و الصائنين و المرقّين للنظام، وغيرهم. وإذا ركزت فقط على مستخدمي نظامك الظاهرين، فقد تنسى حينها بعضاً من أولئك الأشخاص الآخرين المعنيين بالنظام، ويمكن أن يكون ذلك خطيراً جداً! ربما يكون لأولئك المستخدمين حق الفيتو ("لا نستطيع المصادقة على هذا النظام من دون برهنة أن البيانات لم يتم العبث بها")، أو ربما عليهم فرض متطلبات غير وظيفية مهمة، مثل ترقية محددة خلال ١٠ دقائق بالضبط من عمل النظام وترقية من دون إقفال النظام، ... إلخ. وإذا تم إهمال أولئك المستخدمين فلن يتم توثيق هذه الوظائف المهمة في نظامك، وتكون مخاطراً بالانتهاء مع نظام عديم الجدوى.

٢-١-١-٢ تنقية المستخدمين Refining actors

بعد التعمّق في عملية أسر مختلف المستخدمين المتفاعلين مع النظام، وستجد أن بعضهم على علاقة فيما بينهم، كما هو معروض في الشكل رقم (٢-٤).



شكل رقم (٢-٤) إظهار أن المدير هو مستخدم خاص.

إن المستخدم مدير Administrator هو بالفعل نوع خاص من مستخدمي النظام. ويتم استعمال سهم التعميم generalization لإظهار أن أي مدير باستطاعته عمل أي شيء يمكن أن يعمله المستخدم العادي User (مع بعض الخصائص الإضافية). انظر إلى الفصل الخامس للمزيد حول التعميم وسهمه.

٢-١-٢ حالات الاستخدام Use Cases

بعد أسرك لمجموعة أولية من المستخدمين الذين يتفاعلون مع نظامك، يمكنك أن تركب النموذج الصحيح لتلك التفاعلات. والخطوة التالية هي إيجاد حالات يتم استخدام النظام فيها لإنجاز عمل محدد لصالح مستخدم معين، ويطلق على هذه الحالات "حالات الاستخدام". ويمكن تحديد حالات الاستخدام من خلال متطلبات المستخدمين. ويتم

هنا استخلاص مجموعة وظائف واضحة لنظامك من خلال تلك التعريفات المطنبة والغامضة في وثائق متطلبات المستخدم.

وربما تكون حالة الاستخدام عملاً بسيطاً كالسماح للمستخدم بالدخول إلى نظام، أو ربما تكون معقدة كتنفيذ صفقة بشكل موزّع عبر قواعد بيانات متعددة عالمية. إن الأمر المهم تذكره هو أن حالة الاستخدام - من منظور المستخدم - هي استخدام كامل للنظام؛ ويوجد بعض التفاعل مع النظام، كما يوجد بعض الناتج عن ذلك التفاعل، وعلى سبيل المثال، يصف المتطلب أ-١ أحد الاستخدامات الأساسية لنظام إدارة المحتوى CMS: إنشاء حساب مدونة جديد. ويعرض الشكل رقم (٢-٥) كيفية أسر هذا التفاعل كحالة استخدام.



شكل رقم (٢-٥) ترسم حالة الاستخدام في UML على شكل بيضاوي يكتب بداخله اسم يصف التفاعل الذي تمثله.



تذكر أنه إذا كانت حالات الاستخدام هي بالفعل متطلبات، يجب بالتالي أن يكون لديها معايير نجاح أو فشل واضحة جداً. ويجب على المطوّر، والمختبر، والكاتب التقني، والمستخدم معرفة بشكل صريح إذا كان النظام ينجز حالة الاستخدام أم لا.

بعد كل هذا التدرج بالعرض، ربما توقعت أن تكون حالة الاستخدام جزءاً معقداً من الترميز. بالمقابل، وكل ما حصلت عليه هو شكل بيضاوي! إن ترميز حالة الاستخدام بسيط جداً، وغالباً ما يخفي أهميته في أسر الأمور التي يهتم بها النظام. لا تسء فهم ذلك؛ فمن المحتمل

أن تكون حالة الاستخدام هي النموذج الأقوى والوحيد في UML للتأكد من أن نظامك يعمل ما يفترض به عمله.

كيفية إنشاء حالة استخدام جيدة What Makes a Good Use Case؟

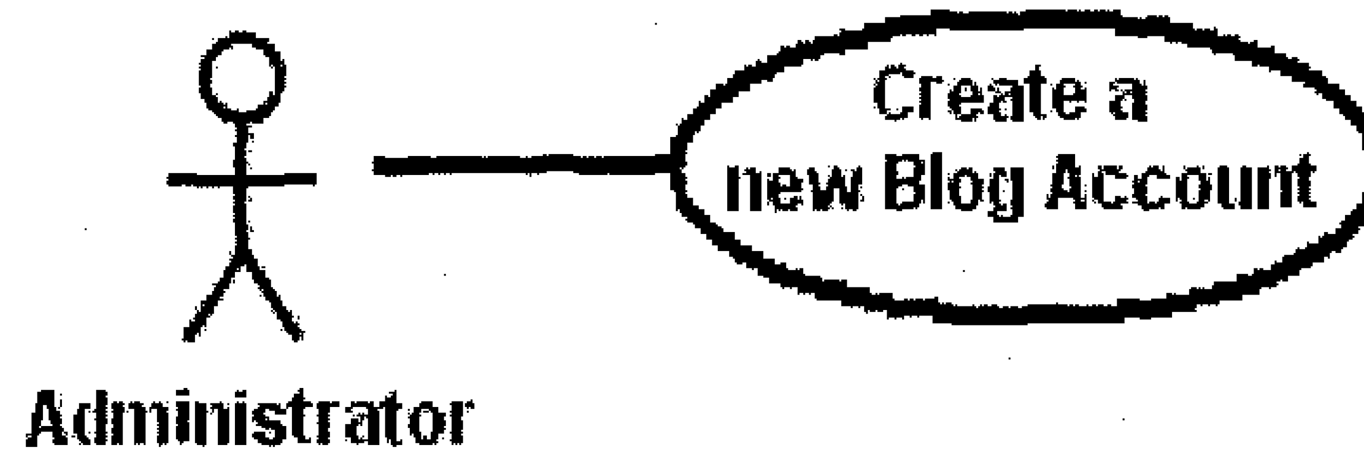
ستساعدك الخبرة في اكتشاف حالات الاستخدام الجيدة وتحديدها، و لكن يوجد طريقة مجرّبة يمكن استعمالها لتحديد ذلك: إن حالة الاستخدام هي الشيء الذي يزوّد مستخدماً أو نظاماً خارجياً ما بنتيجة قابلة للقياس.

أي جزء من سلوك النظام يحقق هذا الاختبار البسيط يمكن على الأرجح أن يكون مرشحاً جيداً ليصبح حالة استخدام.

٢-١-٣ خطوط الاتصال Communication Lines

حتى هذه النقطة، قمنا بتعريف حالات الاستخدام و مستخدمي النظام، ولكن كيف سنربط بينهما لتحدد مثلاً أن المستخدم مدير Administrator معني بحالة الاستخدام "إنشاء حساب مدونة جديد Create a new Blog Account"؟ تأتي هنا مهمة خطوط الاتصال.

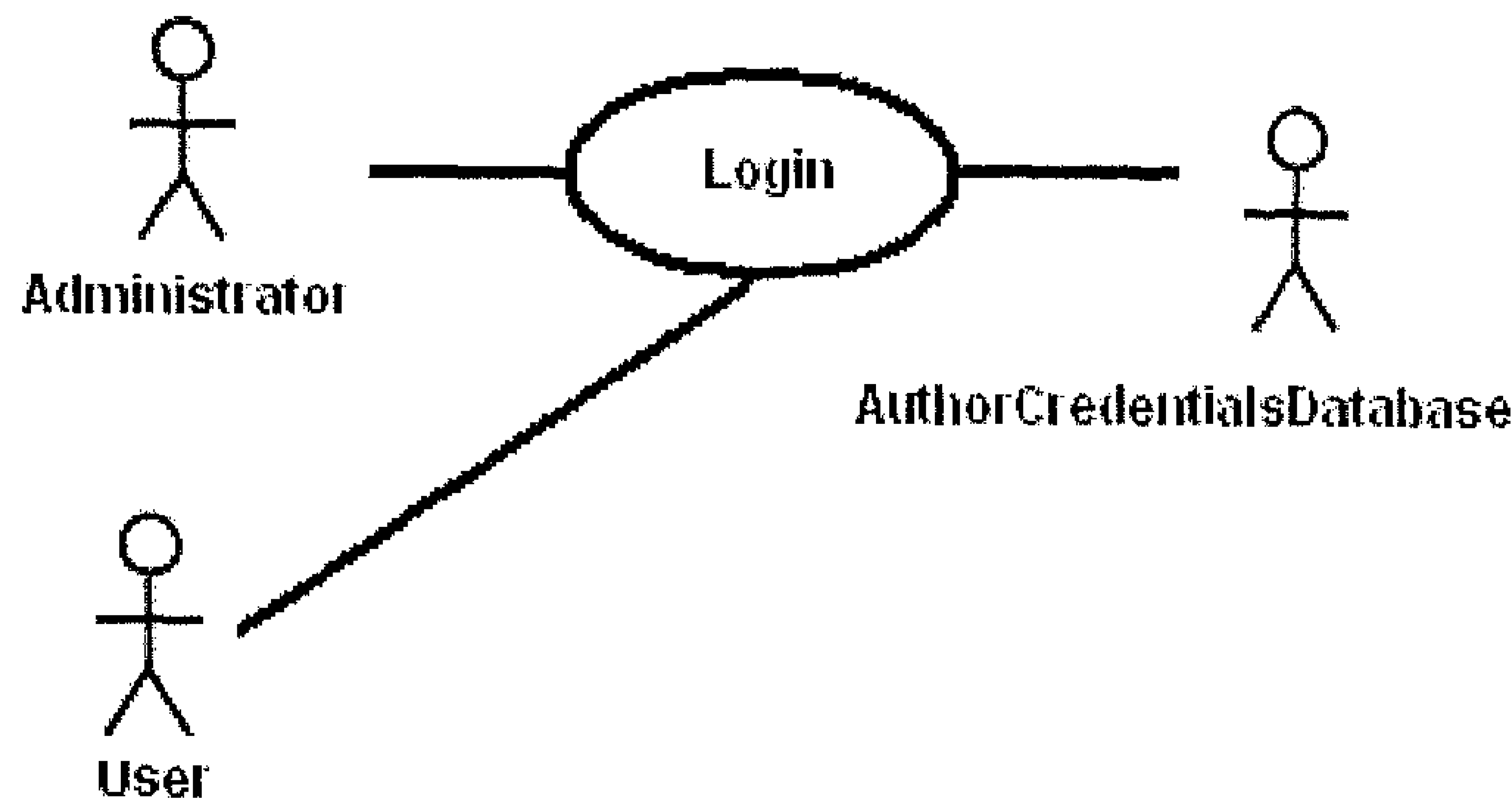
يربط خط الاتصال بين مستخدم و حالة استخدام لإظهار مشاركة المستخدم في حالة الاستخدام. وفي هذا المثال، يشارك المستخدم مدير في حالة الاستخدام "إنشاء حساب مدونة جديد"؛ ويظهر ذلك في الشكل رقم (٢-٦) بإضافة خط اتصال بينهما.



شكل رقم (٢-٦) يربط خط الاتصال المستخدم **Administrator** بحالة الاستخدام "**Create a new Blog Account**"؛ المدير معني بالتفاعل الذي تمثله حالة الاستخدام.

ويظهر هذا المثال البسيط خط اتصال بين مستخدم واحد وحالة استخدام واحدة فقط. ويمكن لأي عدد من المستخدمين المشاركة في حالة استخدام واحدة، وليس هناك نظرياً حدّ معين لهذا العدد.

ولإظهار مجموعة مستخدمين مشاركين في حالة استخدام واحدة، كل ما عليك فعله هو رسم خط اتصال بين هؤلاء المستخدمين والشكل البيضاوي لحالة الاستخدام، كما هو معروض في الشكل رقم (٢-٧).



شكل رقم (٢-٧) تتفاعل حالة الاستخدام "login" أثناء تنفيذها مع ثلاثة مستخدمين.

سيكون أحياناً لمخططات UML خطوط اتصال مع قابلية الملاحة عبرها؛ وعلى سبيل المثال، إذا كان عندنا سهم موجه باتجاه واحد في المخطط، فسيظهر هذا السهم تدفق المعلومات بين المستخدم وحالة الاستخدام، أو يظهر مَنْ يبدأ حالة الاستخدام. ومع أن هذا الترميز مسموح به في اصطلاحات UML، لكنه ليس استعمالاً جيداً لخطوط الاتصال.

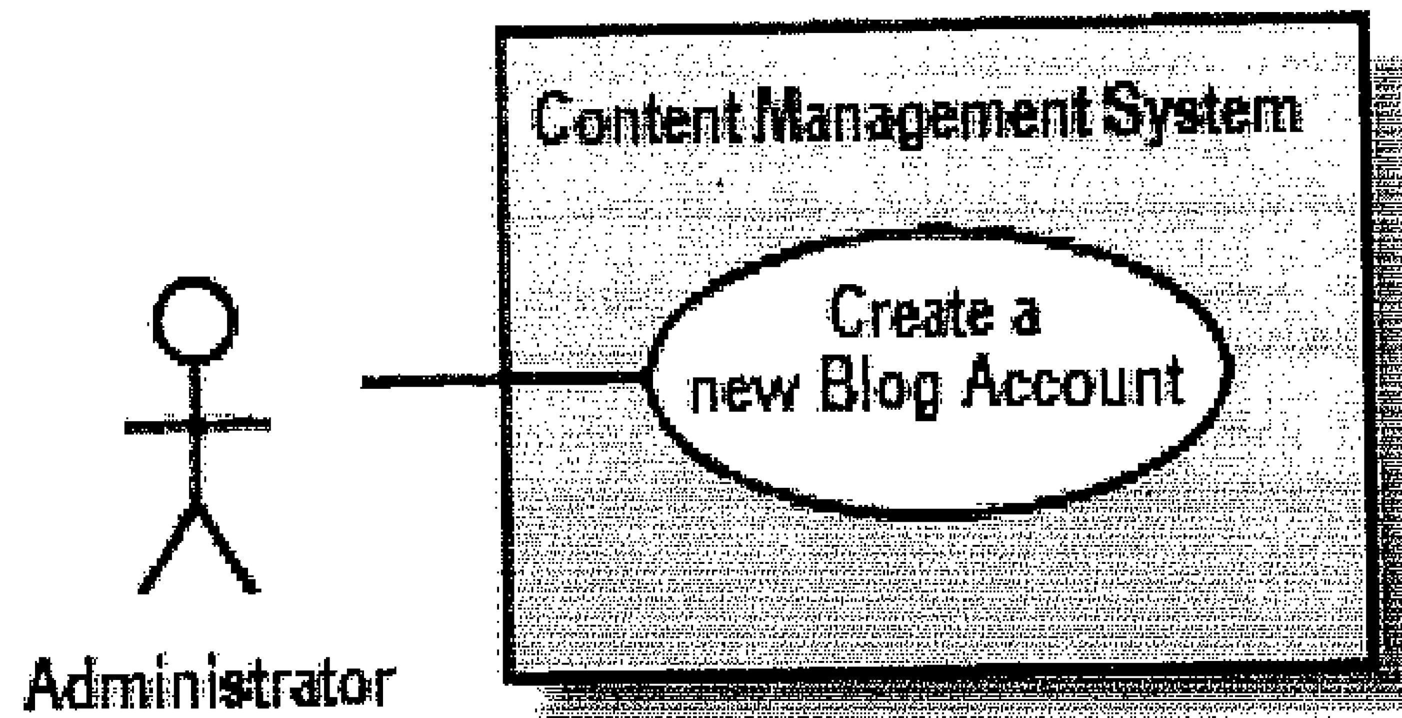
إن الهدف من خط الاتصال هو إظهار أن المستخدم يشارك ببساطة في حالة استخدام، وليس للدلالة على تبادل المعلومات باتجاه محدد ولا أن المستخدم يبدأ حالة الاستخدام. ويكون هذا النوع من المعلومات موجوداً في الوصف المفصل لحالة الاستخدام، ومن هنا ليس هناك معنى لتطبيق

الملاحـة علـى خطـوط الاتصـال. انظـر لاحقـاً في هـذا الفصـل إلى القسـم "توصيفات حالة الاستخدام" للمزيد عن حالات الاستخدام وتوصيفاتها.

٢-١-٤ حدود النظام System Boundaries

يوجـد تفريـق ضـمني بـين المسـتخدمين (خـارج النظـام) وحـالات الاسـتخدام (داخـل النظـام)، ويقـوم بتحديـد حـدود النظـام وتـوفر UML ترميزاً بسيطاً يمكن استخدامه لجعل الأمور شديدة الوضوح.

ولإظهار حدود نظامك على مخطط حالة استخدام، أرسم صندوقاً حـول كـل حـالات الاسـتخدام مـع الاحتفـاظ بالمسـتخدمين خـارج هـذا الصندوق. وتعتبر تسمية صندوقك وفقاً للنظام الذي تطوره ممارسـة جيـدة، كما هو معروض في الشكل رقم (٢-٨) لنظام إدارة المحتوى CMS.



شكل رقم (٢-٨) يقع المستخدم **Administrator** خارج نظام إدارة المحتوى **CMS**، مما يبين بشكل صريح أن حالات الاستخدام يجب أن تقع داخل صندوق حدود النظام، لذا لا يوجد معنى لوجود حالة استخدام خارج حدود النظام.

٢-١-٥ توصيفات حالة الاستخدام Use Case Descriptions

ربما يكون المخطط الذي يعرض حالات الاستخدام والمستخدمين عبارة عن نقطة بداية جيدة، ولكنه لا يزود مصممي النظام بتفاصيل كافية لفهم كيفية إنجاز أعمال النظام بالضبط. وكيف يمكن لمصمم

النظام أن يفهم مَنْ هو المستخدم الأكثر أهمية من خلال ترميز حالة الاستخدام فقط؟ وأي خطوات تنطوي تحت حالة الاستخدام؟ هنا تأتي الوسيلة المثلى للتعبير عن هذه المعلومات المهمة على شكل الوصف النصي، لذلك يجب على كل حالة استخدام أن تكون مرفقة بوصف نصي خاص بها.

ووفقاً للغة النمذجة الموحدة، لا توجد قواعد حازمة و سريعة تحدد ما يدخل بالضبط في وصف حالة الاستخدام، لكن نعرض في الجدول رقم (٢-١) بعض الأمثلة عن أنواع المعلومات بهذا الخصوص.

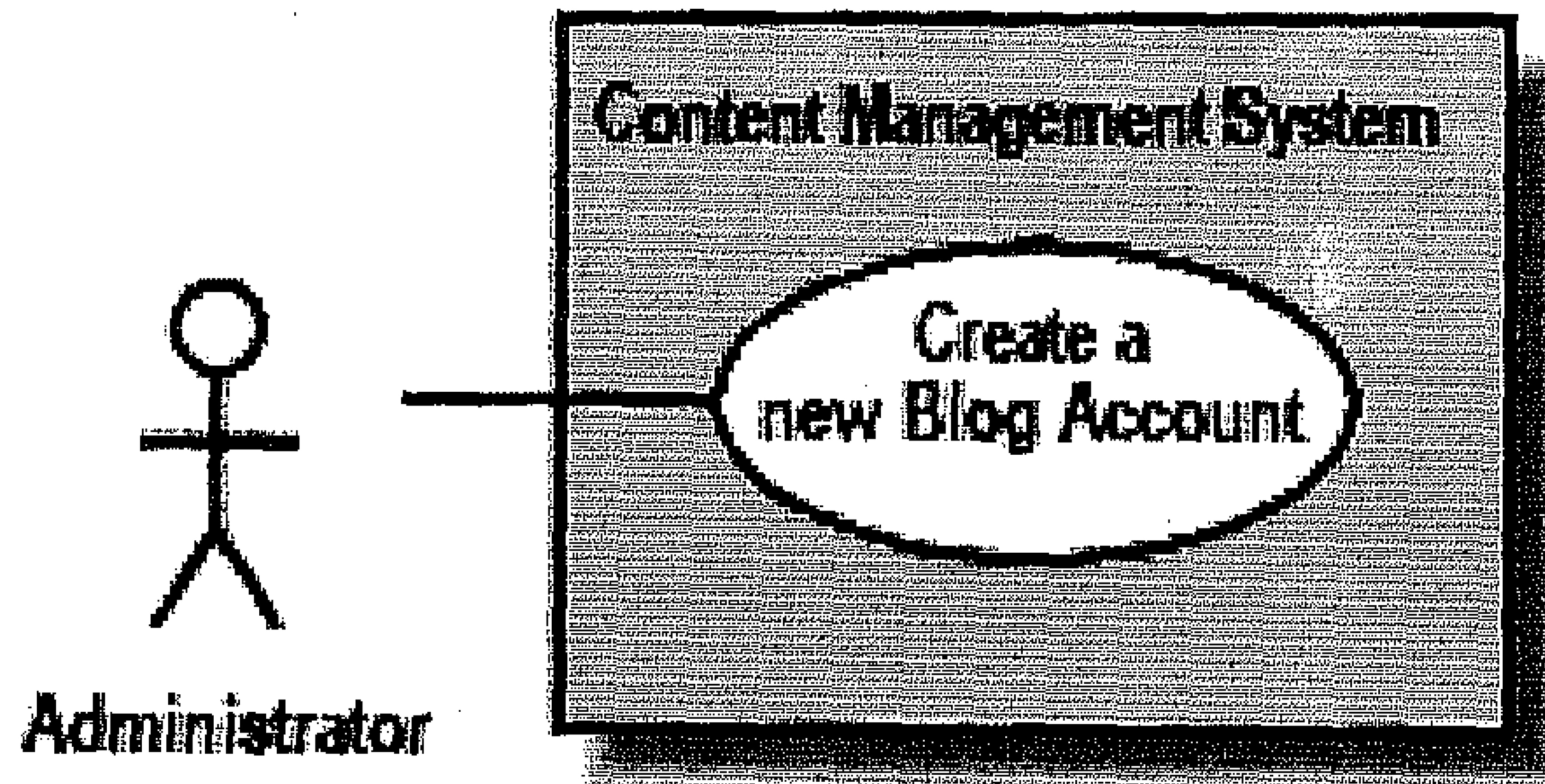
جدول رقم (٢-١) بعض أنواع المعلومات التي يمكن تضمينها في توصيف حالة الاستخدام.

وصف عنصر تفصيلي لحالة الاستخدام	معنى العنصر التفصيلي وسبب فائدته
المتطلبات ذات العلاقة	إشارة إلى المتطلبات التي تنجزها حالة الاستخدام جزئياً أو كلياً.
الهدف ضمن السياق	موقع حالة الاستخدام في النظام و سبب أهميتها.
الشروط المسبقة	ما يستوجب حدوثه قبل أن يصبح بالإمكان تنفيذ حالة الاستخدام.
حالة نهاية ناجحة	كيف يجب أن تكون حالة النظام بعد تنفيذ حالة الاستخدام بنجاح.
حالة نهاية فاشلة	كيف يجب أن تكون حالة النظام إذا فشلت حالة الاستخدام بالتنفيذ بنجاح.
المستخدمون الأساسيون	المستخدمون الرئيسيون المشاركون في حالة الاستخدام. وغالباً ما تضم المستخدمين الذين يطلقون حالة الاستخدام، أو يستلمون مباشرة معلومات من تنفيذها.
المستخدمون الثانويون	المستخدمون المشاركون غير الرئيسيين في تنفيذ حالة الاستخدام.
المُطلق Trigger	الحدث المثار من قبل المستخدم و المسبب تنفيذ حالة الاستخدام.
التدفق الرئيسي	مكان وصف الخطوات المهمة في التنفيذ الطبيعي لحالة الاستخدام.
التوسيعات	وصف لأي خطوات بديلة لتلك الموصوفة في التدفق الرئيسي.

يعرض الجدول رقم (٢-٢) مثالاً لتوصيف حالة الاستخدام "إنشاء حساب مدونة جديد" و الذي يوفر قالباً عملياً لتوصيفاتك الخاصة.

ويعتبر الشكل والمحتوى بالجدول رقم (٢-٢) مجرد مثال، لكن من المفيد تذكّر أن توصيفات حالة الاستخدام والمعلومات التي تحتويها، هي أكثر من مجرد معلومات إضافية مرافقة لمخططات حالة الاستخدام. وفي الحقيقة، إن توصيف حالة الاستخدام يكمّل حالة الاستخدام؛ ومن دون هذا التوصيف لا تكون حالة الاستخدام مفيدة جداً.

لقد كان التوصيف الذي في الجدول رقم (٢-٢) مباشراً إلى حدّ معقول، لكن هناك شيئاً ما غير صحيح تماماً حين تقارن التوصيف بمخطط حالة الاستخدام الأصلي المعروض في الشكل رقم (٢-٩)؛ فرغم أن توصيف حالة الاستخدام يشير إلى مستخدمين اثنين، ويظهر مخطط حالة الاستخدام مستخدماً واحداً فقط.



شكل رقم (٢-٩) من المهم التأكد من تطابق مخططات حالة الاستخدام مع التوصيفات الأكثر تفصيلاً لحالة الاستخدام.

جدول رقم (٢-٢) توصيف كامل لحالة الاستخدام "إنشاء حساب مدونة جديد".

اسم حالة الاستخدام	إنشاء حساب مدونة جديد	
المتطلبات ذات العلاقة	المتطلب أ-١.	
الهدف ضمن السياق	يطلب كاتب جديد أو موجود حساب مدونة جديد من المدير.	
الشروط المسبقة	يكون النظام محدوداً على الكاتبين المعروفين وبالتالي يحتاج الكاتب إلى إثبات مناسب لهويته.	
حالة نهاية ناجحة	قد تم إنشاء حساب مدونة جديد للكاتب.	
حالة نهاية فاشلة	قد تم رفض الطلب المقدم لإنشاء حساب مدونة جديد.	
المستخدمون الأساسيون	المدير.	
المستخدمون الثانويون	قاعدة بيانات اعتماد الكتبة.	
المطلق Trigger	طلب المدير من النظام CMS إنشاء حساب مدونة جديد.	
التدفق الرئيسي	الخطوة	العمل
	١	يطلب المدير من النظام إنشاء حساب مدونة جديد.
	٢	يختار المدير نوعاً معيناً للحساب.
	٣	يدخل المدير تفاصيل الكاتب.
	٤	يتم التحقق من تفاصيل الكاتب باستعمال قاعدة بيانات اعتماد الكتبة.
	٥	يتم إنشاء حساب مدونة جديد.
	٦	يتم إرسال بريد إلكتروني للكاتب يضم ملخصاً عن تفاصيل الحساب الجديد في المدونة.
التوسيعات	الخطوة	العمل المتفرع
	٤- ١	لتأكيد قاعدة بيانات اعتماد الكتبة، وصحة تفاصيل الكاتب
	٤- ٢	يتم رفض الطلب المقدم من الكاتب لإنشاء حساب مدونة جديد.

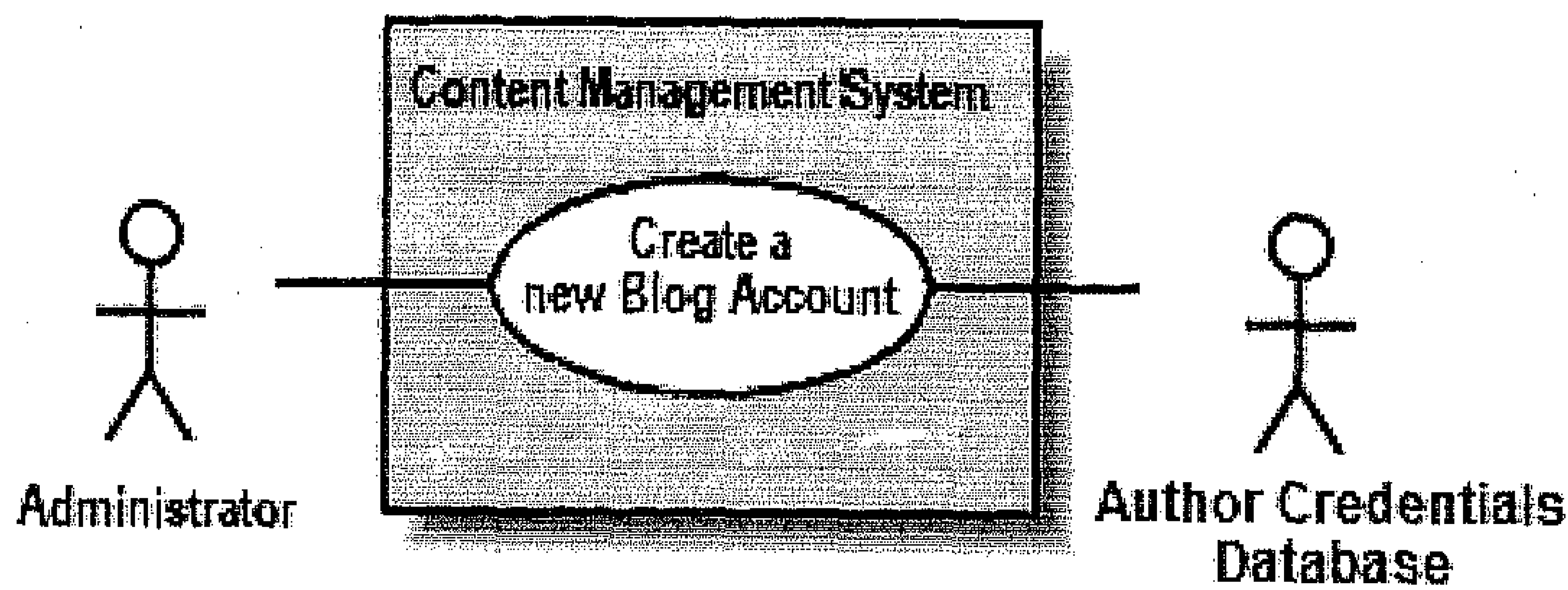


من المفيد مراجعة نموذج حالات الاستخدام مع مستخدمي نظامك إلى أبعد حد ممكن، وذلك لضمان أنك قد أسرت كل الاستخدامات الرئيسية لنظامك ولم تنسَ أياً منها.

لقد ميّز توصيف حالة الاستخدام مستخدماً جديداً، ألا وهو "قاعدة بيانات اعتماد الكتبة". بإنشاء توصيف كامل لحالة الاستخدام "إنشاء حساب مدونة جديد"، وأصبح واضحاً أنه ينقص هذا المستخدم.

كلما أدخلت مزيداً من تفاصيل توصيفات حالات الاستخدام ستجد على الأرجح بعض النقص في عناصر مخططاتك. وينطبق نفس الأمر على أي جانب من نموذجك. وكلما أدخلت تفاصيل أكثر كثر وجوب رجوعك للوراء لتصحيح ما قمت بعمله سابقاً. وهذا ما تدور حوله منهجية التطوير التكراري للنظام Iterative system development. ومع ذلك لا تقلق كثيراً، فإن هذه التنقية والتفصيل لنموذجك هي أمر جيد. ومع كل تكرار في التطوير ستحصل على نموذج أفضل وأكثر دقة لنظامك.

ويعرض الشكل رقم (٢-١٠) المخطط المصحح لحالة الاستخدام بعد إدخال المستخدم الجديد "قاعدة بيانات اعتماد الكتبة".



شكل رقم (٢-١٠) تنقيح مخطط حالة الاستخدام بالتوازي مع توصيف حالة الاستخدام من خلال إضافة قاعدة بيانات اعتماد الكتبة.

كم يجب أن يكون لنموذجك من حالات الاستخدام؟

ليس هناك قاعدة محددة لعدد حالات الاستخدام التي يجب أن يحتويها نموذجك المقترح لنظام ما. ويعتمد عدد حالات الاستخدام على الوظائف التي يجب أن يقوم بها نظامك وفقاً للمتطلبات. وهذا يعني أنه من أجل نظام محدد، فقد تحتاج إلى حالتي استخدام فقط، أو إلى المئات من حالات الاستخدام.

والأكثر أهمية أن يكون عندك حالات الاستخدام الصحيحة، بدلاً من القلق بخصوص كميتها. وكما هو الحال مع أكثر الأشياء في نمذجة النظام، فإن أفضل وسيلة للحصول بشكل صحيح على حالات الاستخدام الخاصة بك هي أن تتعوّد على تطبيقها؛ وستعلمك التجربة ما هو صحيح لأنظمتك الخاصة.

٢-٢ علاقات حالات الاستخدام
Use Case Relationships

تصف حالة الاستخدام الطريقة التي يتصرف بها نظامك لتلبية متطلب ما. وعند تكملة توصيفات حالات الاستخدام، ستلاحظ أن هناك بعض التشابه بين الخطوات في حالات الاستخدام المختلفة. وربما قد تجد أيضاً أن بعض حالات الاستخدام تعمل بعدة أنماط مختلفة أو حالات خاصة. وقد تجد أيضاً حالة استخدام لها تدفقات متعددة أثناء تنفيذها، وسيكون أمراً جيداً إظهار تلك الحالات الاختيارية المهمة على مخططاتك.

أَلاَ يكون أمراً عظيماً إذا تمكنت من التخلص من التكرار في توصيفات حالات الاستخدام وإظهار التدفقات الاختيارية المهمة بشكل صحيح على مخططات حالات الاستخدام؟ ويمكن إظهار سلوك حالة استخدام قابلة لإعادة الاستعمال وحالة استخدام اختيارية وحتى حالة استخدام خاصة بين حالات الاستخدام.

٢-٢-١ العلاقة ((تتضمّن)) The "include" Relationship

حتى الآن، لقد رأيت أن حالات الاستخدام تعمل بشكل نموذجي مع المستخدمين لأسر متطلب ما. وتتمحور العلاقات بين حالات الاستخدام بدرجة كبيرة حول تجزئة سلوك نظامك إلى أجزاء سهلة الإدارة بدلاً من إضافة أي جديد إليه. والهدف من علاقات حالات الاستخدام هو تزويد مصممي نظامك ببعض التوجيهات ذات الطابع المعماري، مما يمكنهم من تجزئة أعمال النظام إلى أجزاء سهلة الإدارة ضمن التصميم المفصّل للنظام.

لنلقي نظرة أخرى على توصيف حالة الاستخدام "إنشاء حساب مدونة جديد" المعروض في الجدول رقم (٢- ٢). يبدو أن هذا التوصيف بسيط بما فيه الكفاية، لكن لنفرض أنه تم إضافة متطلب آخر إلى نظام إدارة المحتوى.



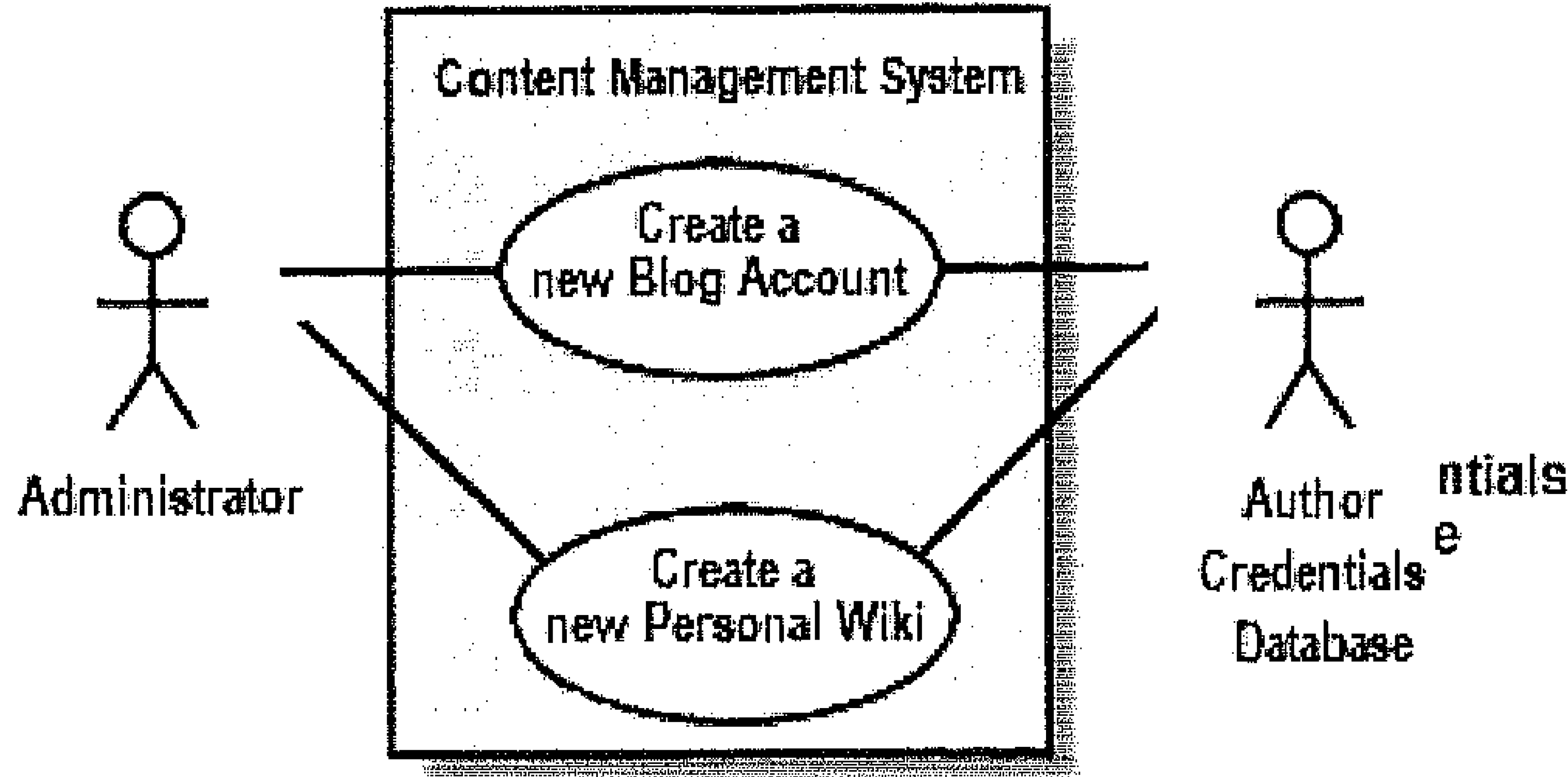
بالإضافة إلى المدونات، يمكن أن يضم نظام إدارة المحتوى CMS أي عدد من الوسائل للعمل على محتواه. إحدى الآليات الرائجة لصيانة الوثائق هي بإنشاء ويكي Wiki. وتسمح الويكيز Wikis للكتبة المتصلين بالإنترنت من إنشاء وتحرير وربط صفحات الوبْ معاً، ومن أجل إنشاء شبكة ذات محتوى مترابط أو ويكي وِبْ Wiki-web. يتوفر مثال مهم عن الويكي في http://www.Wikipedia.org,

المتطلب أ-٢

يجب أن يسمح نظام إدارة المحتوى للمدير بإنشاء حساب ويكي شخصي جديد، شرط أن يتم التحقق من التفاصيل الشخصية للكاتب مقدم الطلب باستعمال قاعدة بيانات اعتماد الكتبة.

لأسر المتطلب أ-٢، نحتاج إلى إضافة حالة استخدام جديدة إلى نظام إدارة المحتوى، كما هو معروض في الشكل رقم (٢-١١).

بما أننا أضفنا حالة الاستخدام الجديدة للنموذج، فقد حان الوقت لتكملة التوصيف المفصل للحالة (المبين في الجدول رقم ٢-٣). إذا احتجت إنعاش ذاكرتك حول معنى العناصر التفصيلية لتوصيف حالة الاستخدام انظر إلى الجدول رقم (٢-١).



شكل رقم (٢- ١١) عادة ما يشير المتطلب الجديد إلى حالة استخدام جديدة للنظام، على الرغم من أن هذا الأمر لا يشكل علاقة واحد مقابل واحد دائماً.

أول ما يجب ملاحظته هو وجود بعض التكرار غير الضروري بين توصيفات حالتي الاستخدام [الجدول رقم (٢-٢) والجدول رقم (٢-٣)]. تحتاج كلتا الحالتَين "إنشاء حساب مدونة جديد" و "إنشاء ويكي شخصي جديد" إلى التحقق من وثائق مقدِّم الطلب. وقد تم بكل بساطة تكرار هذا السلوك في توصيفات الحالتين.

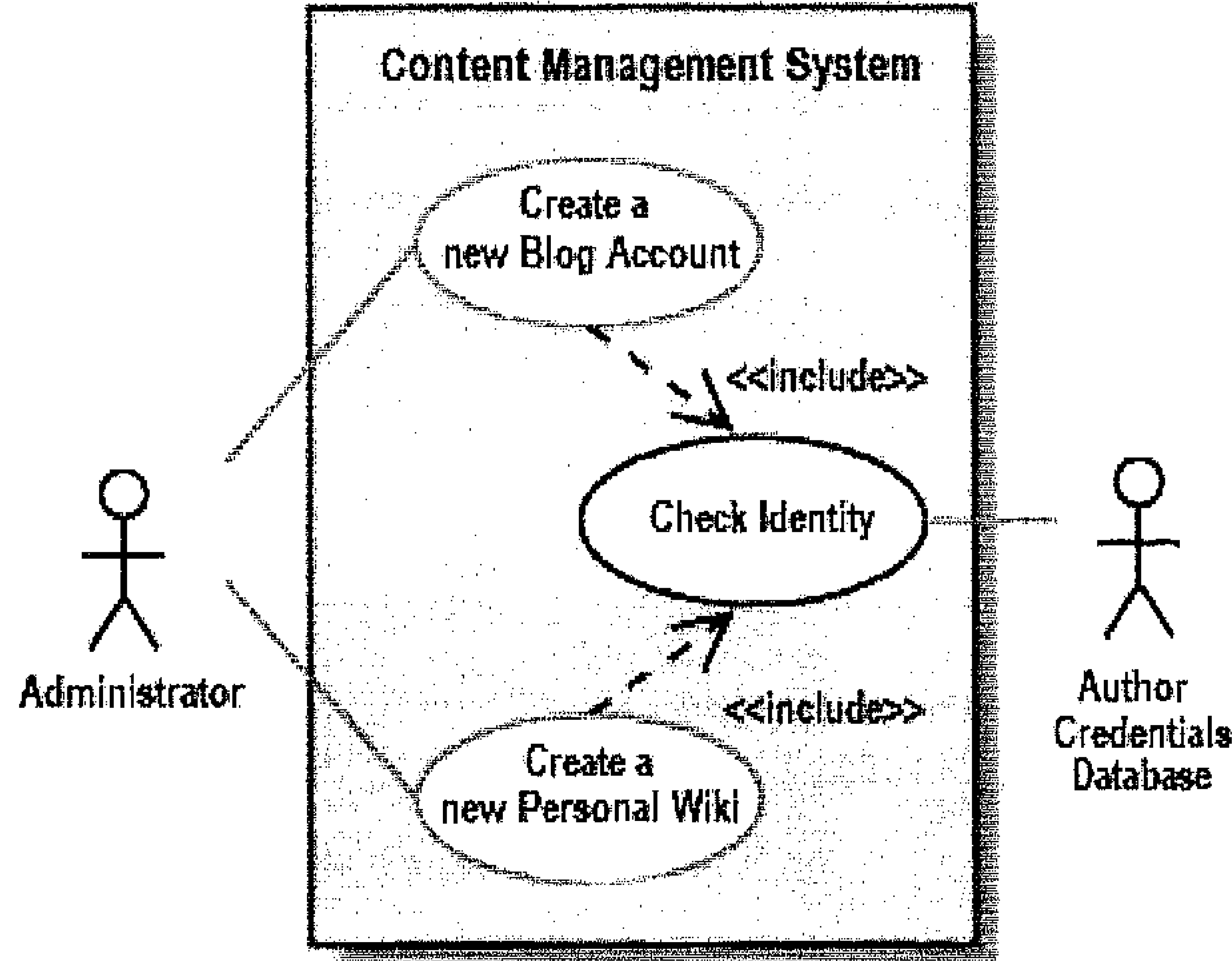
جدول رقم (٢-٣) التوصيف المفصّل لحالة الاستخدام "إنشاء ويكي شخصي جديد".

اسم حالة الاستخدام	إنشاء ويكي شخصي جديد
المتطلبات ذات العلاقة	المتطلب أ- ٢.
الهدف ضمن السياق	يطلب كاتب جديد أو موجود ويكي شخصي جديد من المدير.
الشروط المسبقة	أن يكون للكاتب إثبات مناسب لهويته.
حالة نهاية ناجحة	قد تم إنشاء ويكي شخصي جديد للكاتب.
حالة نهاية فاشلة	قد تم رفض الطلب المقدم لإنشاء ويكي شخصي جديد.

اسم حالة الاستخدام		إنشاء ويكي شخصي جديد
المستخدمون الأساسيون		المدير
المستخدمون الثانويون		قاعدة بيانات اعتماد الكتبة.
المُطلق Trigger		طلب المدير من نظام إدارة المحتوى إنشاء ويكي شخصي جديد.
التدفق الرئيسي	الخطوة	العمل
	١	يطلب المدير من النظام إنشاء ويكي شخصي جديد.
	٢	يدخل المدير تفاصيل الكاتب.
	٣	يتم التحقق من تفاصيل الكاتب باستعمال قاعدة بيانات اعتماد الكتبة.
	٤	يتم إنشاء ويكي شخصي جديد .
	٥	يتم إرسال بريد إلكتروني للكاتب يضم ملخصاً عن تفاصيل الويكي الشخصي الجديد.
التوسيعات	الخطوة	العمل المتفرع
	٣- ١	لا تؤكد قاعدة بيانات اعتماد الكتبة صحة تفاصيل الكاتب.
	٣- ٢	رفض الطلب المقدم من الكاتب لإنشاء ويكي شخصي جديد.

يمكن أسر السلوك التكراري المشترك بين الحالتين السابقتين بشكل أفضل ووضعه في حالة استخدام جديدة كلياً. ويمكن عندئذ استعمال العلاقة «تتضمّن include»، إعادة استعمال حالة الاستخدام الجديدة من قِبَل الحالتين "إنشاء حساب مدونة جديد" و "إنشاء ويكي شخصي جديد" (كما هو معروض في الشكل رقم ٢-١٢).

وتعني العلاقة <<include>> المرافقة للسهم المنقط أن حالة الاستخدام الخارج منها السهم تعيد بالكامل استعمال كل خطوات حالة الاستخدام المتضمّنة عن الطرف الآخر للسهم. يبين الشكل رقم (٢-١٢) إن حالتي الاستخدام "إنشاء حساب مدونة" و "إنشاء ويكي شخصي جديد" تعيد بالكامل استعمال كل الخطوات المُعلن عنها في حالة الاستخدام "التحقق من الهوية Check Identity".



شكل رقم (٢-١٢) تدعم العلاقة <<include>> إعادة الاستعمال بين حالات الاستخدام.

ويمكنك أيضاً ملاحظة أن حالة الاستخدام "التحقق من الهوية" التي في الشكل رقم (٢-١٢) لم تُربط مباشرة بالمستخدم مدير؛ فهو يحقق هذا الارتباط من خلال حالات الاستخدام التي تتضمّنها. وعلى أية حال، إن حالة الاستخدام "التحقق من الهوية" تملك حصرياً رابط الاتصال الذي بينها وبين المستخدم "قاعدة بيانات اعتماد الكتبة". ويفيد هذا التغيير بالتأكيد على أن حالة الاستخدام "التحقق من الهوية" هي الوحيدة التي تعتمد مباشرة على رابط يصلها بالمستخدم "قاعدة بيانات اعتماد الكتبة".

لإظهار العلاقة <<include>> في توصيفات حالة الاستخدام، تحتاج إلى إزالة الخطوات الزائدة الموجودة في توصيفات حالتي الاستخدام"إنشاء حساب مدونة جديد" و "إنشاء ويكي شخصي جديد"، و استعمال بدلاً منها الحقل "الحالات المُتضمّنة Included Cases" و التركيبة:

"تتضمن :: <اسم حالة الاستخدام>" "**include::<use case name>**"

وذلك للإشارة إلى حالة الاستخدام حيث توجد الخطوات التي أعيد استعمالها، كما هو معروض في الجدولين رقم (٢-٤) و (٢-٥).

جدول رقم (٢-٤) إظهار العلاقة <<تتضمّن>> في توصيف حالة استخدام محددة باستعمال حقل الحالات المتضمنة و التركيبة "تتضمّن:: <اسم حالة الاستخدام>

اسم حالة الاستخدام	إنشاء حساب مدونة جديد	
المتطلبات ذات العلاقة	المتطلب أ- ١.	
الهدف ضمن السياق	يطلب كاتب جديد أو موجود حساب مدونة جديد من المدير.	
الشروط المسبقة	أن يكون للكاتب إثبات مناسب لهويته.	
حالة نهاية ناجحة	تم إنشاء حساب مدونة جديد للكاتب.	
حالة نهاية فاشلة	تم رفض الطلب المقدم لإنشاء حساب مدونة جديد.	
المستخدمون الأساسيون	المدير	
المستخدمون الثانويون	لا أحد	
المُطلق Trigger	طلب المدير من نظام إدارة المحتوى إنشاء حساب مدونة جديد.	
الحالات المتضمنة	التحقق من الهوية.	
التدفق الرئيسي	الخطوة	العمل
	١	يطلب المدير من النظام إنشاء حساب مدونة جديد.
	٢	يختار المدير نوع حساب.
	٣	يدخل المدير تفاصيل الكاتب.
	٤ تتضمن : التحقق من الهوية	قد تم التحقق من تفاصيل الكاتب.
	٥	قد تم إنشاء الحساب الجديد.
	٦	قد تم إرسال بريد إلكتروني للكاتب يضم ملخصاً عن تفاصيل حساب المدونة الجديد.

جدول رقم (٢-٥) يكسب توصيف حالة الاستخدام "إنشاء ويكي شخصي جديد" بعض التحسين أيضاً.

اسم حالة الاستخدام	إنشاء ويكي شخصي جديد	
المتطلبات ذات العلاقة	المتطلب أ- ٢.	
الهدف ضمن السياق	يطلب كاتب جديد أو موجود إنشاء ويكي شخصي جديد من المدير.	
الشروط المسبقة	أن يكون للكاتب إثبات مناسب لهويته.	
حالة نهاية ناجحة	تم إنشاء ويكي شخصي جديد للكاتب.	
حالة نهاية فاشلة	تم رفض الطلب المقدم لإنشاء ويكي شخصي جديد.	
المستخدمون الأساسيون	المدير.	
المستخدمون الثانويون	لا أحد.	
المُطلق Trigger	يطلب المدير من نظام إدارة المحتوى إنشاء ويكي شخصي جديد.	
الحالات المتضمنة	التحقق من الهوية.	
التدفق الرئيسي	الخطوة	العمل
	١	يطلب المدير من النظام إنشاء ويكي شخصي جديد.
	٢	يدخل المدير تفاصيل الكاتب.
	٣ تتضمن: التحقق من الهوية	تم التحقق من تفاصيل الكاتب.
	٤	تم إنشاء ويكي شخصي جديد.
	٥	تم إرسال بريد إلكتروني للكاتب يضم ملخصاً عن تفاصيل الويكي الشخصي الجديد.

يمكنك الآن إنشاء توصيف حالة استخدام للخطوات القابلة لإعادة الاستعمال ضمن حالة الاستخدام "التحقق من الهوية"، كما هو معروض في الجدول رقم (٢-٦).

جدول رقم (٢-٦) توصيف حالة الاستخدام "التحقق من الهوية" المحتوي للخطوات القابلة لإعادة الاستعمال.

اسم حالة الاستخدام	التحقق من الهوية	
المتطلبات ذات العلاقة	المتطلب أ- ١، المتطلب أ- ٢.	
الهدف ضمن السياق	حاجة فحص تفاصيل كاتب و التحقق منها بشكل دقيق.	
الشروط المسبقة	أن يكون للكاتب الذي تم التحقق منه إثبات مناسب لهويته.	
حالة نهاية ناجحة	قد تم التحقق من التفاصيل بنجاح.	
حالة نهاية فاشلة	فشل التحقق من التفاصيل.	
المستخدمون الأساسيون	قاعدة بيانات اعتماد الكتبة.	
المستخدمون الثانويون	لا أحد.	
المُطلق Trigger	تزويد النظام بوثائق الكاتب للتحقق منها.	
التدفق الرئيسي	الخطوة	العمل
	١	تم تزويد التفاصيل إلى النظام.
	٢	تتحقق قاعدة بيانات اعتماد الكتبة من التفاصيل.
	٣	تم إرجاع التفاصيل على أنه تم التحقق منها من قبل قاعدة بيانات اعتماد الكتبة.
التوسيعات	الخطوة	العمل المتفرع
	٢- ١	عدم تأكيد قاعدة بيانات اعتماد الكتبة صحة التفاصيل.
	٢- ٢	إرجاع التفاصيل على أنها غير صحيحة.

لماذا الأسرار على مفهوم إعادة الاستعمال بين حالات الاستخدام؟ لماذا نبقي على الخطوات المتماثلة بشكل منفصل داخل حالات الاستخدام؟ إن لإعادة الاستعمال هنا فائدتين مهمتين؛ وهما:

- يزيل مفهوم إعادة الاستعمال من خلال العلاقة <<تتضمّن>> الحاجة إلى عمليات القص و اللصق بين توصيفات حالات الاستخدام، لأن التحديثات تتم في مكان واحد فقط بدلاً من إجرائها في كل حالة استخدام.

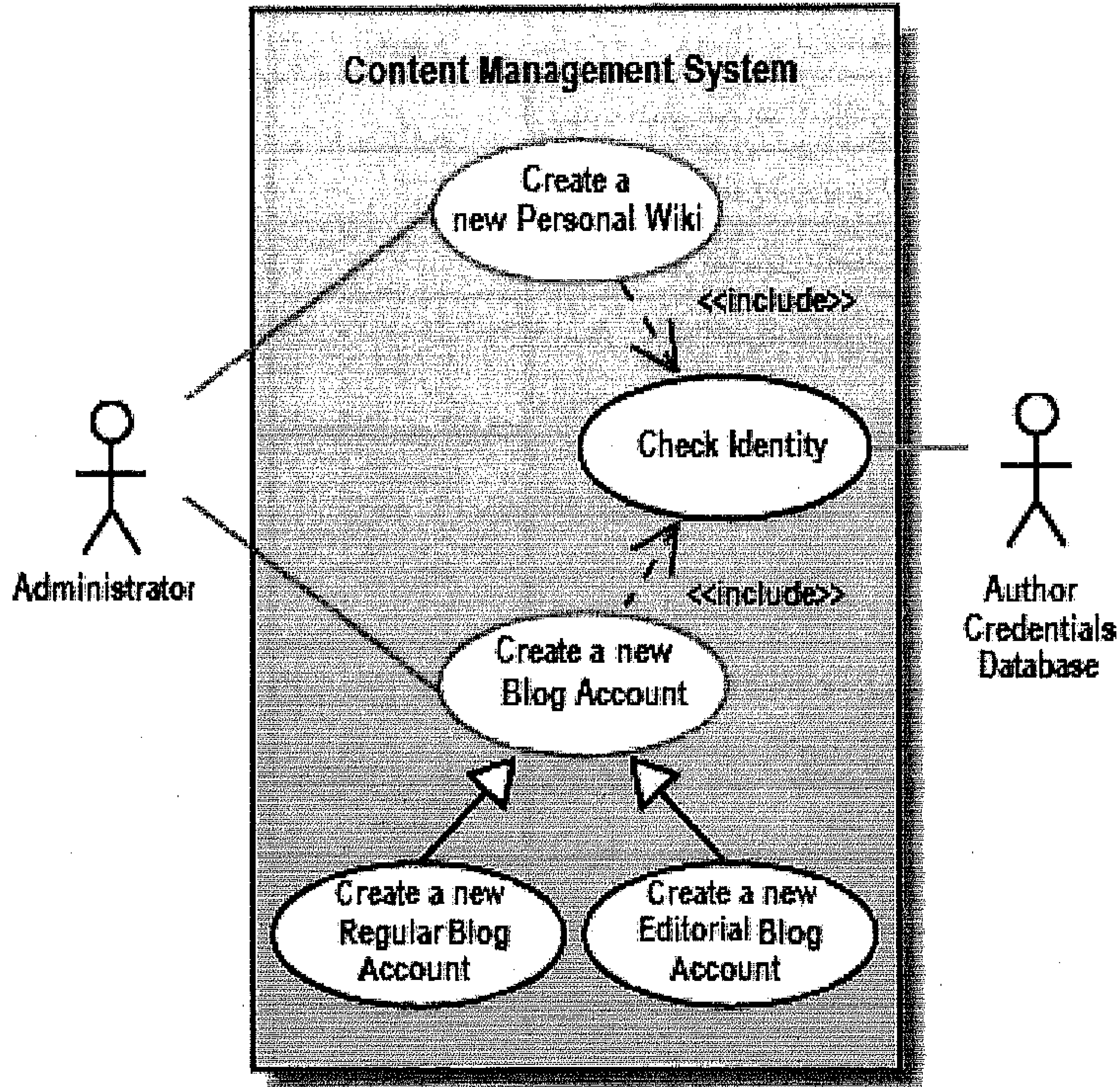
• تشير العلاقة <<تتضمّن>> وقت تصميم النظام بوضوح إلى حاجة إنجاز "التحقق من الهوية" على شكل جزءٍ قابلٍ لإعادة الاستعمال في نظامك.

٢-٢-٢ الحالات الخاصة Special Cases

يصادف أحياناً بعد التدقيق بتفاصيل حالة استخدام ما أنه يمكن تطبيقها على عدة حالات مختلفة ولكن مع إجراء بعض التغيرات الطفيفة. على عكس العلاقة <<تتضمّن>> التي تسمح بإعادة استعمال مجموعة سلوكية فرعية صغيرة، ويمكن إجراء ذلك بتطبيق حالة استخدام مع تغيرات طفيفة لمجموعة من الوضعيات المحددة. باستخدام مصطلاحات التوجه الكائني، ربما يكون لدينا عدد من الحالات المُخصّصة specialized لحالة استخدام مُعمّمة generalized.

ولنوضح الفكرة من خلال المثال التالي: يحتوي نظام إدارة المحتوى على حالة استخدام واحدة "إنشاء حساب مدونة جديد" التي تصف الخطوات المتطلبة لإنشاء حساب محدد. لكن: ماذا لو كان نظام إدارة المحتوى يدعم عدة أنواع مختلفة من الحسابات في المدونة، وتختلف الخطوات المتطلبة لإنشاء كل من هذه الأنواع من الحسابات بشكل طفيف عن حالة الاستخدام الأصلية؟ تريد أن تصف السلوك العام general لإنشاء حساب مدونة - الذي تم أسره في حالة الاستخدام "إنشاء حساب مدونة جديد" - ومن ثم تعريف حالات الاستخدام المخصّصة التي تسمح بإنشاء حسابات من أنواع خاصة، مثل حساب عادي مع مدونة واحدة أو حساب تحريري مع إمكانية إجراء تغييرات في مداخل مجموعة من المدونات.

مـن هنـا يـأتي مفهـوم التعمـيم بـين حالـة الاسـتخدام use case generalization. هناك أسلوب أكثر شيوعاً للإشارة إلى مفهوم التعميم من خـلال اسـتعمال مصـطلح الوراثـة inheritance. تكـون الوراثـة لحـالات الاستخدام مفيدة عندما نريد تبيان أن حالة استخدام ما هي نوع خاص من حالة استخدام أخرى. لإظهار الوراثة بين حالات الاستخدام، قم باستعمال سهم التعميم لربط حالة الاستخدام الأكثر تعميماً، أو الأهل parent، بحالة الاستخدام الأكثر تخصيصاً (اتجاه السهم من الخاص إلى العام). يعرض الشكل رقم (٢-١٣) كيف يمكن توسيع حالات استخدام نظام إدارة المحتوى لتبيان أنه بالإمكان إنشاء نوعين مختلفين من الحسابات في المدونة.



شكل رقم (٢-١٣) يمكن إنشاء نوعين من الحسابات في المدونة من قبل نظام الإدارة، حساب عادي **regular** وحساب تحريري **editorial**.

جدول رقم (٢-٧) يمكننا استعمال حقل حالات الاستخدام الأساسي ضمن التوصيف المفصل لإظهار حالة استخدام خاصة لحالة استخدام أكثر عمومية.

اسم حالة الاستخدام	إنشاء حساب مدونة تحريري جديد	
المتطلبات ذات العلاقة	المتطلب أ- ١.	
الهدف ضمن السياق	يطلب كاتب موجود أو جديد إنشاء حساب مدونة تحريري جديد من المدير.	
الشروط المسبقة	أن يكون للكاتب إثبات مناسب لهويته.	
حالة نهاية ناجحة	تم إنشاء حساب مدونة تحريري جديد للكاتب.	
حالة نهاية فاشلة	تم رفض طلب إنشاء حساب مدونة تحريري جديد.	
المستخدمون الأساسيون	المدير.	
المستخدمين الثانويين	لا أحد.	
المُطلق Trigger	طلب المدير من نظام إدارة المحتوى إنشاء حساب تحريري جديد يسمح للكاتب بتحرير التدوينات في مجموعة مدونات.	
حالات الاستخدام الأساسية	إنشاء حساب مدونة جديد.	
التدفق الرئيسي	الخطوة	العمل
	١	يطلب المدير من النظام إنشاء حساب مدونة جديد.
	٢	يختار المدير النوع تحريري للحساب.
	٣	يدخل المدير تفاصيل الكاتب.
	٤	يختار المدير المدونات التي يكون للحساب عليها حقوق التحرير.
	٥ تتضمن: التحقق من الهوية	تم التحقق من تفاصيل الكاتب.
	٦	تم إنشاء الحساب التحريري الجديد.
	٧	تم إرسال بريد إلكتروني للكاتب يضم ملخصاً عن تفاصيل الحساب التحريري الجديد.
التوسيعات	الخطوة	العمل المتفرع
	٥- ١	غير مسموح للكاتب بتحرير المدونات المحددة.
	٥- ٢	تم رفض طلب إنشاء حساب مدونة تحريري.
	٥- ٣	تم تسجيل رفض الطلب على شكل جزء من سجل عمليات الكاتب.

لننظر بدقة أكثر على توصيف حالة الاستخدام المخصّصة "إنشاء حساب مدونة تحريري"، يمكننا ملاحظة كيف تم إعادة استعمال مجمل سلوكيات حالة الاستخدام الأعمّ "إنشاء حساب مدونة جديد". و نحتاج إلى إضافة التفاصيل الخاصة بإنشاء حساب تحريري جديد فقط (انظر الجدول رقم ٢-٧).

إن وراثة أو تعميم حالة الاستخدام هي وسيلة قوية لإعادة استعمال حالات الاستخدام، لذلك علينا تحديد الخطوات الإضافية المطلوبة في حالات الاستخدام المخصصة فقط (انظر إلى الفصل الخامس للمزيد من المعلومات عن الوراثة بين الأصناف classes).

يجب الحذر عند استعمال الوراثة، لأننا قلنا إنه يجب على كل خطوة في حالة الاستخدام العامة أن تحدث فعلياً في حالات الاستخدام المخصّصة. بالإضافة إلى أن كل علاقة لحالة الاستخدام العامة مع مستخدمين خارجيين أو حالات استخدام أخرى، مثل العلاقة <<تتضمّن>> بين حالة الاستخدام العامة"إنشاء حساب مدونة جديد" المتضمنة لحالة الاستخدام "التحقق من الهوية"، يجب أن يكون لها معنى لحالات الاستخدام الأكثر تخصيصاً أيضاً، كحالة الاستخدام "إنشاء حساب مدونة تحريري جديد".

إذا لم ترد حقاً أن تنفذ حالة الاستخدام الأكثر تخصيصاً كل ما تصفه حالة الاستخدام العامة، فلا تستعمل علاقة التعميم. و بدلاً من ذلك، يمكنك استعمال إما العلاقة <<تتضمّن **include**>> المعروضة في القسم السابق أو العلاقة <<توسّع **extend**>> المعروضة في القسم التالي.

٢-٢-٣ العلاقة <<توسّع>> The <<extend>> Relationship

يجب التحذير قبل أي تفسير للحاشية <<توسّع>> بأنها النوع الأكثر جدلاً وصعوبة بين علاقات حالات الاستخدام. ويظهر أن علاقة حالة الاستخدام <<توسّع>> هي الأقل فهماً أو الأصعب للتداول بشكل صحيح داخل مجتمع UML، وهذا يظهر مشكلة صغيرة عندما تحاول التعلم عنها. يعرض الشكل رقم (٢-١٤) كيف تعمل العلاقة <<توسّع>>؛ ألق نظرة عليه ثم دعنا نتعمق في بعض مفاهيم و نظريات UML.



شكل رقم (٢-١٤) يوجد شبه قليل بين العلاقة <<توسّع>> و العلاقة <<تتضمّن>> و لكنه ينتهي هنا.

بالنسبة لمبرمجي لغة جافا، تبدو العلاقة <<توسّع>> من النظرة الأولى شبيهة جداً بالوراثة بين الأصناف classes. ويمكن في لغة جافا توسيع صنف انطلاقاً من صنف أساسي. وبشكل مشابه، ويمكن في لغة C++ ولغة C# التصريح عن الوراثة بين الأصناف، وغالباً ما نقول بأن صنفاً ما يُوسِّع صنفاً آخر. وفي كلتا الحالتين، تكون علاقة التوسيع بين الأصناف تعني علاقة الوراثة. وبالتالي من الطبيعي للمبرمجين أن تعني علاقة <<توسّع>> شيئاً ما كالوراثة.

لقد رأيت في القسم السابق كيف تصرح حالات الاستخدام عن الوراثة باستعمال سهم التعميم، لِمَ الحاجة إلى نوع آخر من الأسهم مع الحاشية <<توسّع>>؟ وهل سهم التعميم والحاشية <<توسّع>> تعنيان

نفس الشيء؟ لسوء الحظ، تتشارك الحاشية <<توسّع>> الشيء القليل جداً مع الوراثة، وبالتالي فإنهما لا تعنيان الشيء نفسه.

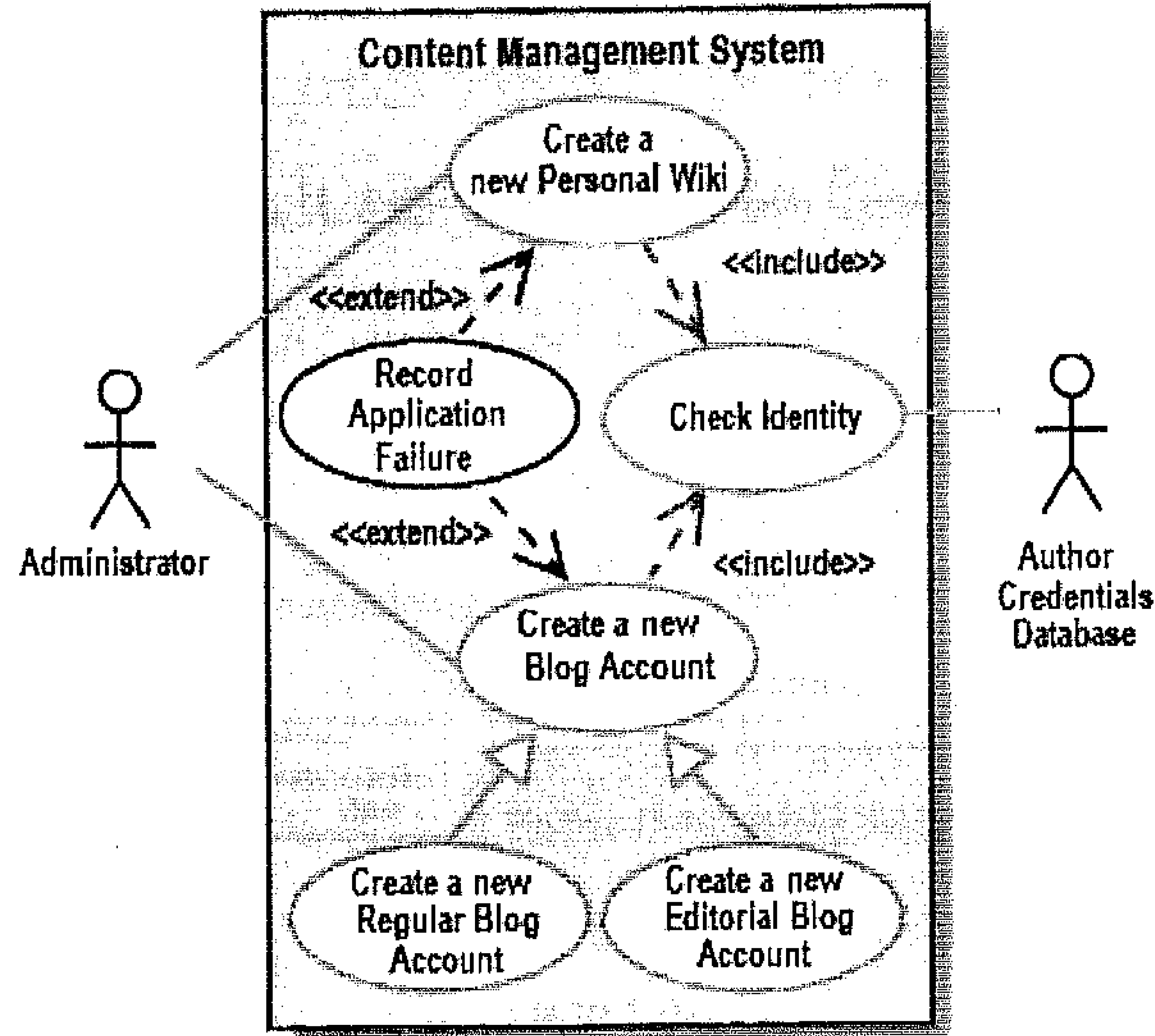
اختار مصممو UML 2.0 رؤية مختلفة جداً لمعنى العلاقة <<توسّع>> بين حالات الاستخدام. لقد أرادوا توفير وسيلة لإعادة استعمال سلوك حالة استخدام بالكامل، بصفة مشابه لعلاقة <<تتضمّن>>، ولكن أن تكون إعادة الاستعمال هذه اختيارية وتعتمد إما على وقت التشغيل أو على قرارات تطبيق النظام.

في مثال نظام إدارة المحتوى CMS، قد تريد حالة الاستخدام "إنشاء حساب مدونة جديد" تسجيل رفض طلب إنشاء حساب من قبل كاتب جديد، وإضافة هذه المعلومات إلى السجل التاريخي لطلبات الكاتب. ويمكن إضافة الخطوات الإضافية إلى توصيف حالة الاستخدام "إنشاء حساب مدونة جديد" لإظهار هذا السلوك الاختياري، كما تبينه الخطوة ٣-٤ في الجدول رقم (٢-٨).

ويفيد السلوك المأسور في الخطوة ٣-٤ أيضاً في حال رفض العميل الحساب لسبب ما أثناء تنفيذ حالة الاستخدام "إنشاء حساب ويكي شخصي جديد". وفقاً للمتطلبات، ويكون هذا السلوك القابل لإعادة الاستعمال اختيارياً في كلتا الحالتين؛ لا تريد تسجيل رفض ما في حال قبول طلب حساب مدونة جديد أو حساب ويكي شخصي. تكون العلاقة <<توسّع>> مثالية في هذا النوع من حالات إعادة الاستعمال، كما هو معروض في الشكل رقم (٢-١٥).

جدول رقم (٢-٨) يمكن العثور على السلوك المرشح لإعادة استعمال العلاقة <<توسّع>> في قسم التوسيعات لتوصيف حالة الاستخدام.

اسم حالة الاستخدام	إنشاء حساب مدونة جديد	
المتطلبات ذات العلاقة	المتطلب أ- ١.	
الهدف ضمن السياق	يطلب كاتب موجود أو جديد حساب مدونة جديد من المدير.	
الشروط المسبقة	أن يكون للكاتب إثبات مناسب لهويته.	
حالة نهاية ناجحة	تم إنشاء حساب مدونة جديد للكاتب.	
حالة نهاية فاشلة	تم رفض الطلب لإنشاء حساب مدونة جديد.	
المستخدمون الأساسيون	المدير.	
المستخدمون الثانويون	لا أحد.	
المُطلق Trigger	طلب المدير من نظام إدارة المحتوى إنشاء حساب مدونة جديد.	
الحالات المتضمنة	التحقق من الهوية	
التدفق الرئيسي	الخطوة	العمل
	١	يطلب المدير من النظام إنشاء حساب مدونة جديد.
	٢	يختار المدير نوع الحساب.
	٣	يدخل المدير تفاصيل الكاتب.
	٤ تتضمن: التحقق من الهوية	تم التحقق من تفاصيل الكاتب.
	٥	تم إنشاء الحساب الجديد.
	٦	تم إرسال بريد إلكتروني للكاتب يضم ملخصاً عن تفاصيل حساب المدونة الجديد.
التوسيعات	الخطوة	العمل المتفرع
	٤-١	غير مسموح للكاتب بإنشاء مدونة جديدة.
	٤-٢	تم رفض طلب حساب المدونة.
	٤-٣	تم تسجيل رفض الطلب كجزء من السجل التاريخي للكاتب.



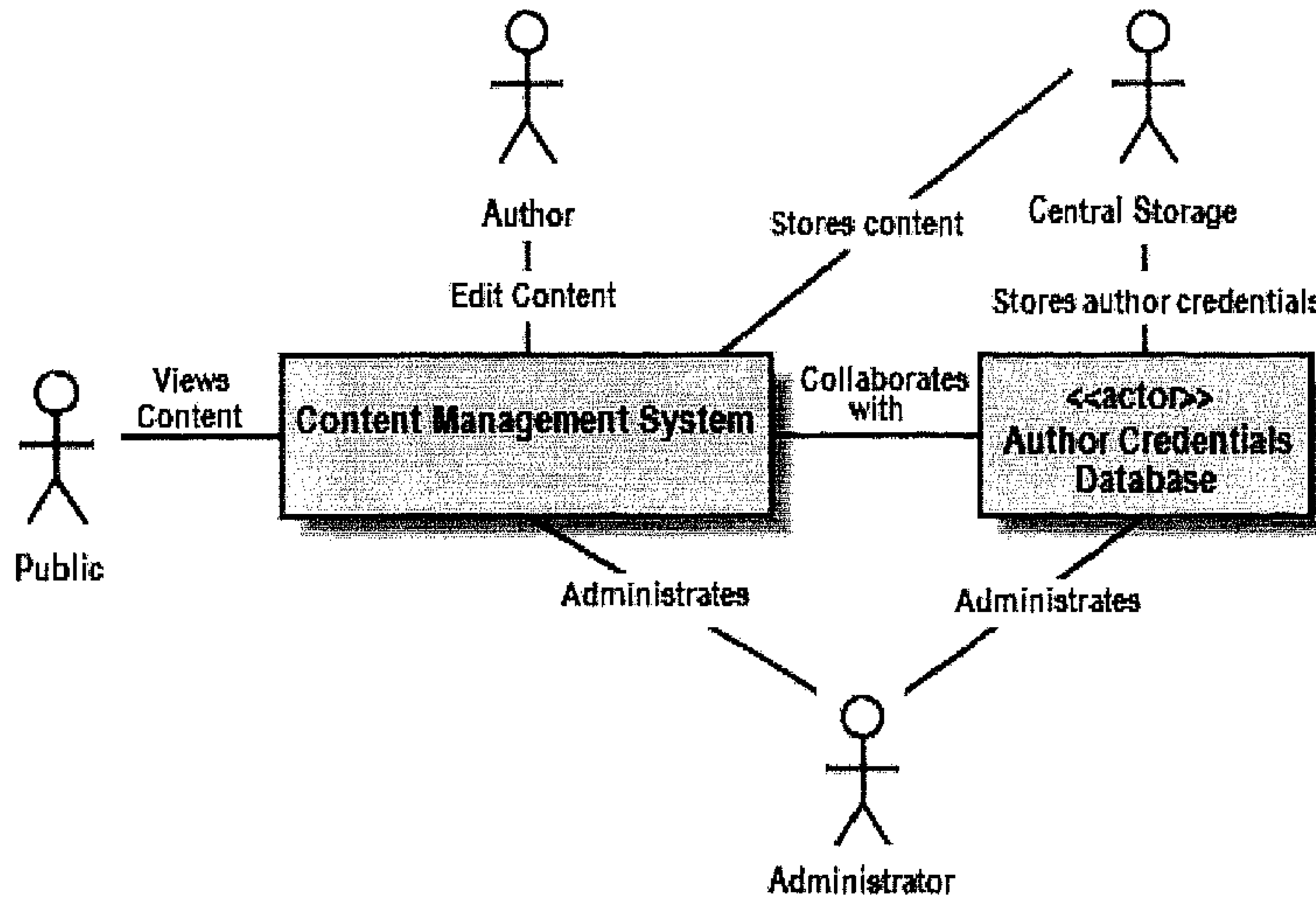
شكل رقم (٢-١٥) تؤدي العلاقة <<توسّع>> دوراً في إظهار إمكانية المشاركة العرضية في سلوك تسجيل رفض الطلب، وذلك ضمن حالتي الاستخدام "إنشاء حساب ويكي شخصي جديد" و "إنشاء حساب مدونة جديد".

إن حالة الاستخدام الجديدة "تسجيل فشل الطلب Record Application Failure" (كما يعني اسمها) تأسر كل السلوكيات المرتبطة بتسجيل فشل طلب كاتب سواء كان لويكي شخصي أو لنوع حساب مدونة محدد. باستعمال العلاقة <<توسّع>>، ويعاد استعمال سلوك حالة الاستخدام "تسجيل فشل الطلب" بشكل اختياري من قبل حالتي الاستخدام "إنشاء حساب مدونة جديد" و "إنشاء حساب ويكي شخصي جديد" في حال تم رفض الطلب.

٢-٣ مخططات ملخص حالة الاستخدام
Use Case Overview Diagrams

عند محاولة فهم نظام ما، من المفيد الحصول على نظرة مقتضبة عن السياق الذي يدور حوله النظام. لهذا الغرض، تزود UML مخطط

ملخص حالة الاستخدام. وتتيح مخططات ملخص حالة الاستخدام فرصة رسم صورة واسعة لسياق أو مجال النظام (يقدم الشكل رقم (٢-١٦) مثالاً عن ذلك).



شكل رقم (٢-١٦) مخطط ملخص حالة استخدام يعرض سياق نظام إدارة المحتوى.

لسوء الحظ، تم تسمية ملخصات حالة الاستخدام بشكل سيء كأنها لا تحتوي على أي حالات استخدام. ولا تعرض حالات الاستخدام بسبب تصميم الملخص لتوفير سياق للنظام؛ ولا يكون داخل النظام المعبر عنه بحالات الاستخدام مرئي عادة.

وتعتبر ملخصات حالة الاستخدام مكاناً مفيداً لعرض أي قصاصات إضافية من المعلومات عند فهم مكان النظام في العالم. وغالباً ما تتضمّن هذه القصاصات العلاقات وخطوط الاتصال بين المستخدمين. ولا تحتوي عادة هذه الأجزاء السياقية من المعلومات كثيراً من التفصيل، وهي تشكل بدرجة أكبر مكان احتواء ونقطة بداية لبقية تفاصيل النموذج.

٢-٤ ما هي الخطوة التالية؟

بالرغم من أن هذا الكتاب، مثل لغة النمذجة الموحدة، لا يروّج لأي عملية تطوير للأنظمة، وقد تم أخذ بعض الخطوات المشتركة بعد تكملة أول دراسة لحالات الاستخدام.

بعد الحصول على نموذج لحالات الاستخدام، يكون الوقت مناسباً عادة للبدء بالتنقيب عن النشاطات عالية المستوى التي يجب على النظام تنفيذها لإنجاز حالات استعماله. (انظر الفصل الثالث للحصول على معلومات عن مخططات النشاط).

وبعد استيعاب النشاطات عالية المستوى بشكل جيد، انظر إلى الأصناف والمكوّنات التي ستكوّن فعلياً أجزاء نظام. وربما يكون عندك بعض الأفكار عمّا تحتويه تلك الأصناف، وبالتالي من الطبيعي أن تكون المحطة التالية هي إنشاء بضعة مخططات أصناف أولية. (انظر الفصل الرابع للحصول على معلومات عن مخططات الأصناف).

وبغض النظر عن الخطوة القادمة، لا يعني مجرد الحصول على نموذج لحالات الاستخدام أنك قد انتهيت تماماً من حالات الاستخدام. ولا يوجد شيء ثابت في الحياة، وينطبق هذا بالتأكيد على متطلبات النظام. وبما أن المتطلب يتغير - إمّا بسبب اكتشاف بعض القيود الجديدة للنظام أو بسبب تغيير المستخدم لرأيه - فتحتاج للرجوع إلى الوراء لتنقيح حالات الاستخدام، وذلك للتأكّد من أنك ما زلت تطوّر النظام الذي يريده المستخدمون.

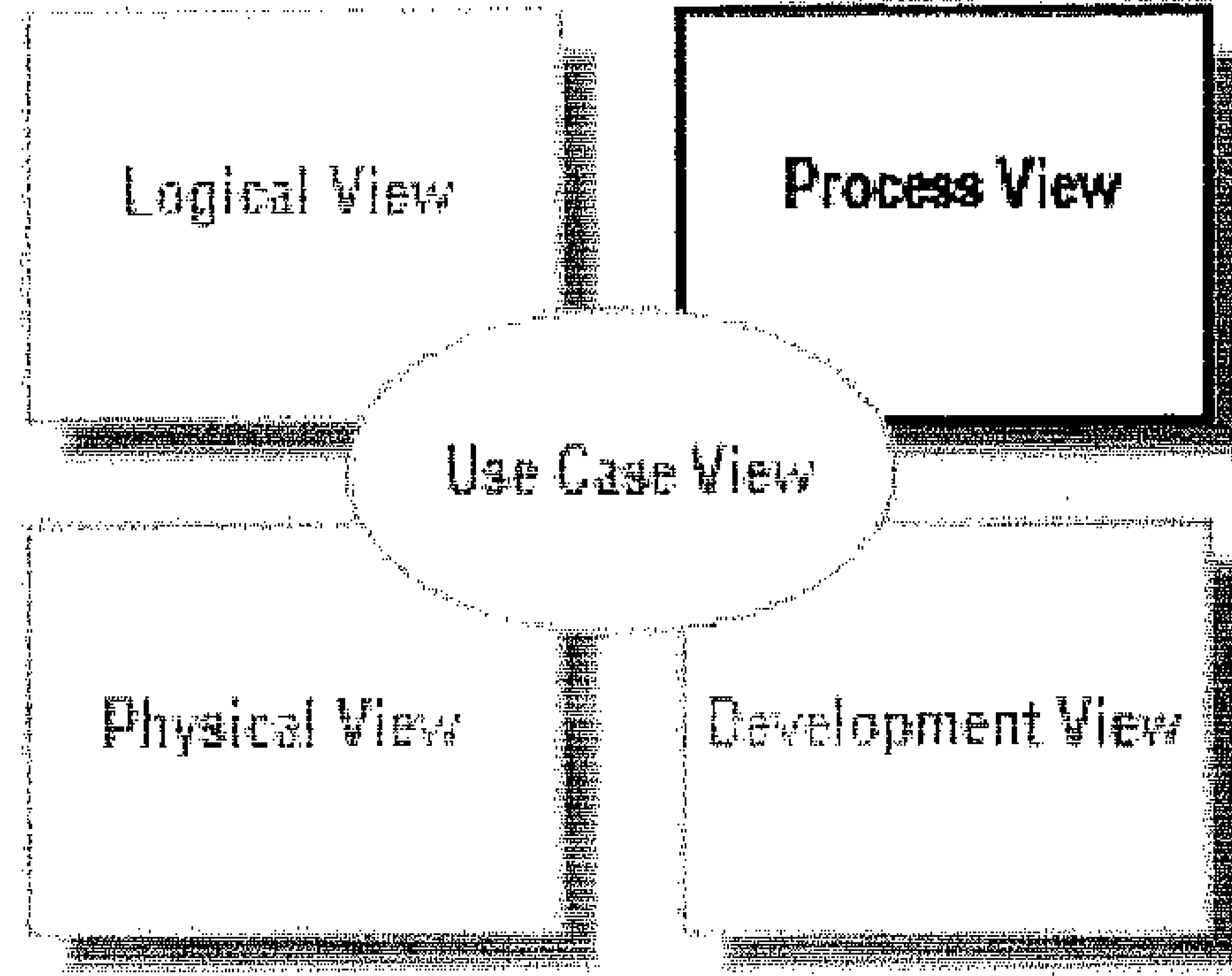
نمذجة تدفقات عمل الأنظمة: مخططات النشاط

MODELING SYSTEM WORKFLOWS: ACTIVITY DIAGRAMS

تظهر حالات الاستخدام "ماذا **what**" يجب أن يعمله النظام. وتسمح لك مخططات النشاط بتحديد "كيف **how**" سينجز النظام أهدافه. وتُظهِر مخططات النشاط الأعمال عالية المستوى المرتبطة معاً لتمثيل عملية تحدث في النظام. وعلى سبيل المثال، يمكنك استعمال مخططات النشاط لنمذجة الخطوات المتعلقة بإنشاء حساب مدونة.

وتفيد مخططات النشاط بشكل جيد خصوصاً في نمذجة عمليات الأعمال **business processes**. وتتألف عملية العمل من مجموعة مهام منسّقة تحقق هدف العمل، مثل شحن طلبات الزبائن. وتسمح بعض أدوات إدارة عمليات الأعمال Business Process Management (BPM) بتعريف عمليات الأعمال باستعمال مخططات النشاط أو ترميزات رسومية مماثلة، ومن ثم تنفيذها. على سبيل المثال، يسمح هذا بتعريف عملية مصادقة على تسديد مبلغ وتنفيذها، حيث تستدعي إحدى هذه الخطوات خدمة وِبْ للمصادقة على بطاقة الائتمان، وذلك باستعمال ترميز رسومي سهل، مثل مخططات النشاط.

إن مخطط النشاط هو مخطط UML الوحيد في منظور العملية لنموذج النظام، كما هو معروض في الشكل رقم (٣-١).



شكل رقم (٣-١) يظهر منظور العملية (عمليات النظام عالية المستوى) وهذا ما تجيد عمله مخططات النشاط بالضبط.

إن مخططات النشاط واحدة من المخططات المألوفة بكثرة في UML، لأنها تستعمل رموزاً مشابهة لترميزات المخطط الانسيابي flowchart المعروفة جداً؛ ولذا فهي مفيدة لوصف العمليات لجمهور واسع. وفي الحقيقة، فإن لمخططات النشاط جذوراً في المخططات الانسيابية Flowchart، بالإضافة إلى مخطط الحالة في UML، و مخططات تدفق البيانات Data Flow Diagram، وشبكات Petri.

٣-١ أساسيات مخطط النشاط
Activity Diagram Essentials

دعنا ننظر إلى العناصر الأساسية لمخطط النشاط، من خلال نمذجة العملية التي صادفناها مؤخراً في الكتاب، أي خطوات حالة الاستخدام "إنشاء حساب مدونة". يحتوي الجدول رقم (٣-١) [الجدول

(٢-١) في الأصل] على توصيف حالة الاستخدام "إنشاء حساب مدونة جديد". ويصف القسمان "التدفق الرئيسي" و "التوسيعات" الخطوات في عملية إنشاء حساب مدونة.

جدول رقم (٣-١) توصيف حالة الاستخدام "إنشاء حساب مدونة جديد".

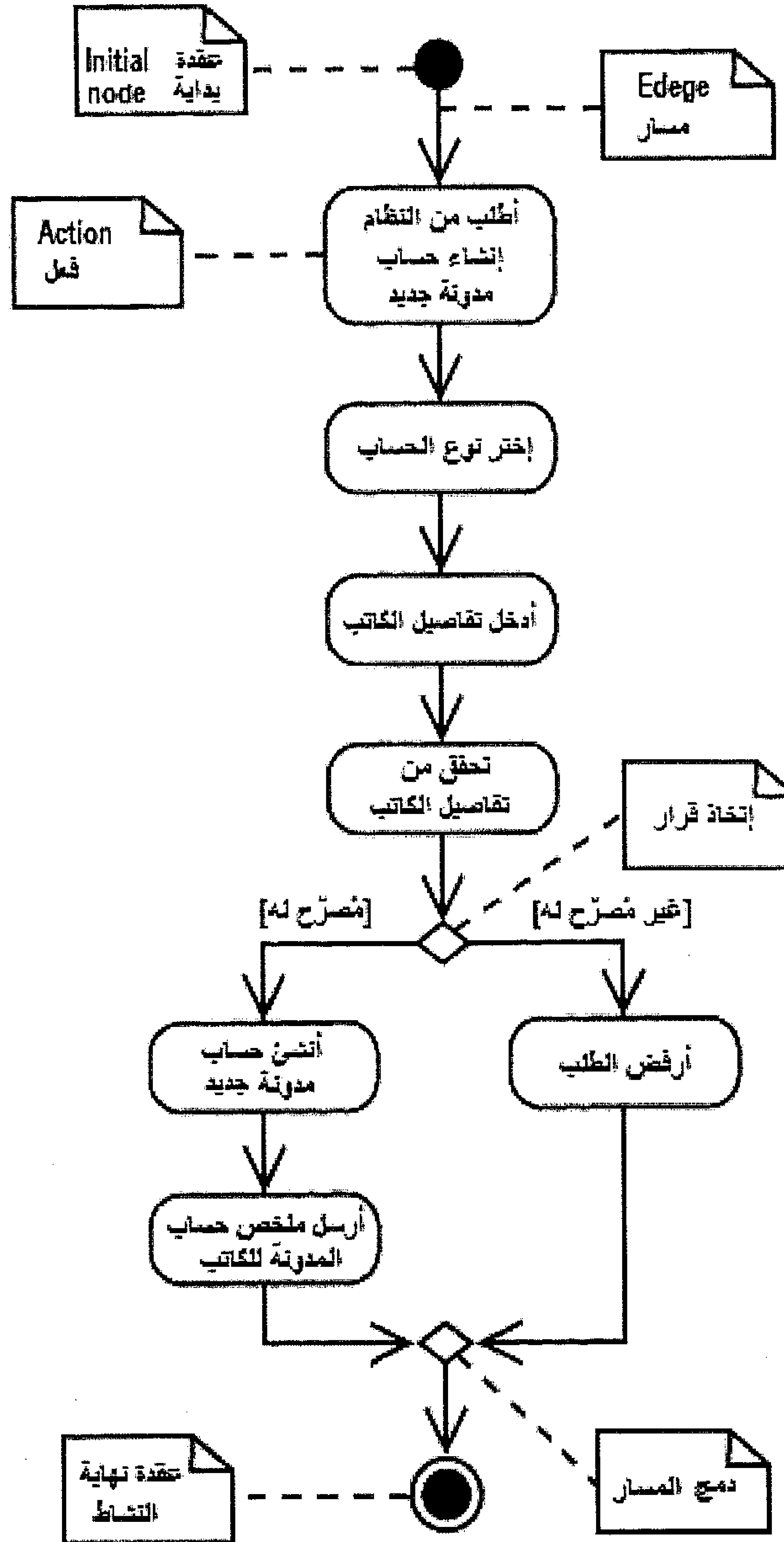
اسم حالة الاستخدام	إنشاء حساب مدونة جديد	
المتطلبات ذات العلاقة	المتطلب أ- ١.	
الهدف ضمن السياق	يطلب كاتب جديد أو موجود حساب مدونة جديد من المدير.	
الشروط المسبقة	يكون النظام مقصوراً على الكتبة الذين تم التعرّف عليهم وبالتالي يحتاج الكاتب أن يكون عنده إثبات مناسب لهويته.	
حالة نهاية ناجحة	تم إنشاء حساب مدونة جديد للكاتب.	
حالة نهاية فاشلة	تم رفض الطلب المقدم لإنشاء حساب مدونة جديد.	
المستخدمون الأساسيون	المدير.	
المستخدمون الثانويون	قاعدة بيانات اعتماد الكتبة.	
المُطلق Trigger	يطلب المدير من النظام إنشاء حساب مدونة جديد.	
التدفق الرئيسي	الخطوة	العمل
	١	يطلب المدير من النظام إنشاء حساب مدونة جديد.
	٢	يختار المدير نوع حساب.
	٣	يدخل المدير تفاصيل الكاتب.
	٤	يتم التحقق من تفاصيل الكاتب باستعمال قاعدة بيانات اعتماد الكتبة.
	٥	تم إنشاء حساب مدونة جديد.
	٦	تم إرسال بريد إلكتروني للكاتب يضم ملخصاً عن تفاصيل حساب المدونة الجديد.
التوسيعات	الخطوة	العمل المتفرع
	٤- ١	لا تتحقق قاعدة بيانات اعتماد الكتبة من تفاصيل الكاتب.
	٤- ٢	تم رفض الطلب المقدم من الكاتب لإنشاء حساب مدونة جديد.

يبين الشكل رقم (٣-٢) ترميز مخطط النشاط لعملية إنشاء حساب مدونة. ويفيد مخطط النشاط هنا؛ لأنه يساعد في تصوُّر خطوات حالة الاستخدام بشكل أفضل (مقارنة بترميز الجدول في توصيف حالة الاستخدام)، وبشكل خاص خطوات التفرّع التي تعتمد على صحة التحقق من الكاتب.

وفي الشكل رقم (٣-٢)، يبدأ النشاط بعقدة بداية **initial node** مرسومة كدائرة معبّأة. وتحدد العقدة الابتدائية بداية النشاط. وفي الطرف الآخر للمخطط ترسم عقدة نهاية النشاط **activity final node** كدائرتين مركزيتين مع دائرة داخلية معبّأة، وهي تحدد نهاية النشاط.

وتكون الأمور التي بين عقدتي بداية ونهاية النشاط عبارة عن أفعال **actions** يتم رسمها مثل مستطيلات مدوّرة الزوايا. إن الأفعال هي الخطوات المهمة التي تحدث في النشاط العام، مثل اختر نوع الحساب **Select Account Type**، وأدخل تفاصيل الكاتب **Enter Author's Details**، وهكذا. ويمكن أن يكون الفعل عبارة عن سلوك تم إنجازه، أو عملية حسابية، أو أي خطوة أساسية في العملية.

ويعرض تدفق النشاط باستعمال خطوط موجهّة (أسهم) تسمى مسارات **edges or paths**. ويحدد رأس السهم الذي على طرف النشاط اتجاه التدفق من فعل إلى الذي يليه مباشرة. ويسمى الخط الداخل في العقدة: مسار قادم **incoming edge**، ويسمى الخط الخارج من العقدة: مسار خارج **outgoing edge**. وتربط الخطوط الأفعال معاً لتحدد تدفق النشاط العام: أولاً، تصبح عقدة البداية نشطة، ثم ينشط "أطلب من النظام إنشاء حساب مدونة جديد"، وهكذا.



شكل رقم (٣-٢) تنمذج مخططات النشاط السلوك الديناميكي مع التركيز على العمليات؛ عرض العناصر الأساسية لمخطط النشاط عملية "إنشاء حساب مدونة".

تسمى العقدة الأولى التي على شكل معيّن بعقدة قرار **decision**، مقارنة مع التعليمة الشرطية if-then-else في البرمجة. لاحظ أنه يوجد مسارا خروج من عقدة القرار في الشكل رقم (٣-٢)، كل منها معنون بتعابير شرطية. يتم اتباع مسار خروج واحد فقط من عقدة القرار حسب التعبير الشرطي إذا كان الكاتب مُصرّح له أم لا. وتسمى العقدة الثانية التي بشكل المُعيّن عقدة دمج **merge**، حيث تقوم بدمج المسارات التي تتفرع من عقدة قرار لتجعلها مساراً واحداً، وتقوم بتحديد نهاية السلوك الشرطي.

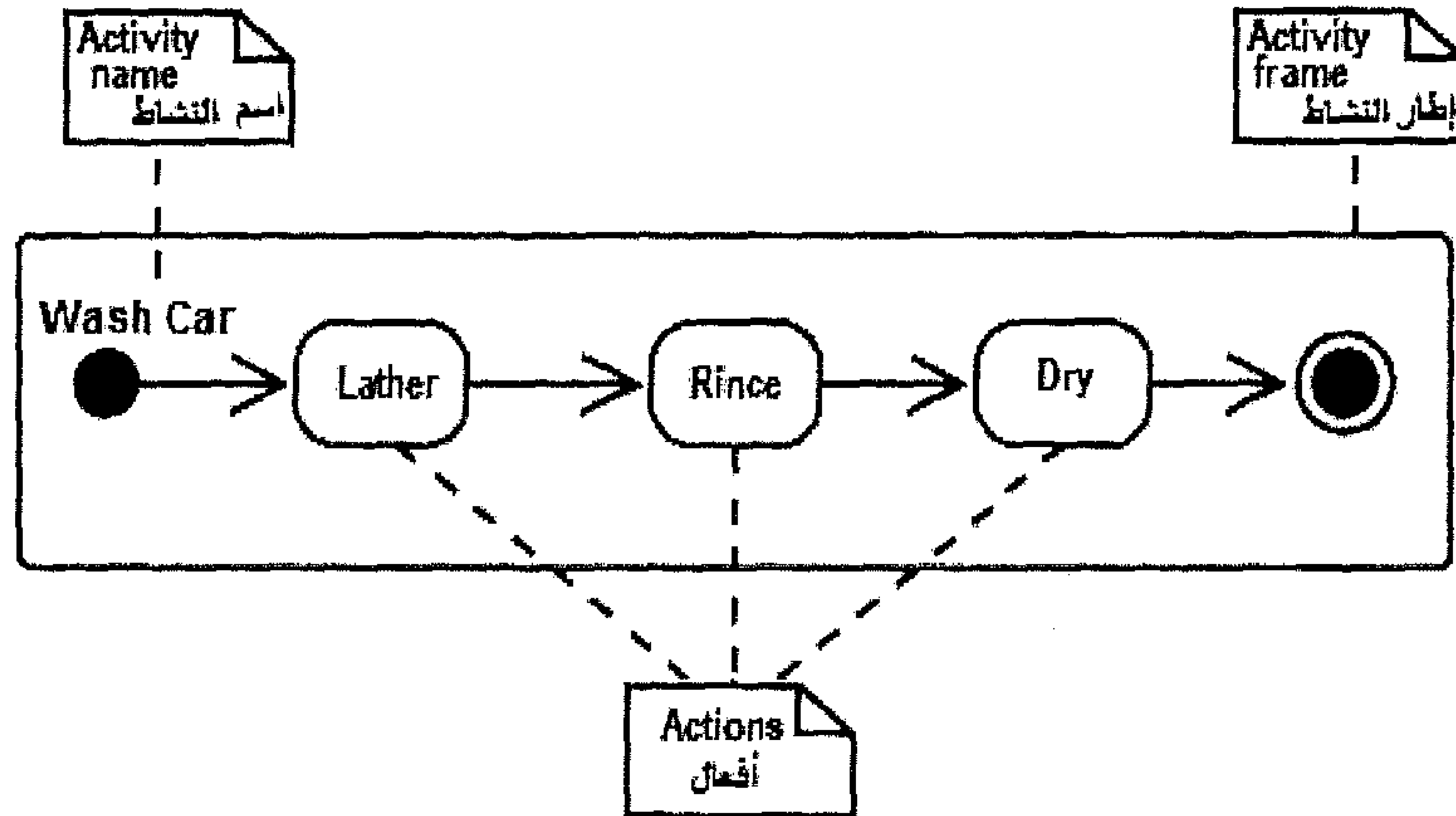
لقد تم ذكر الكلمة "تدفق flow" عدة مرات سابقاً، وقد تسأل ما يعني التدفق؟ يعتمد الجواب على سياق الاستعمال للكلمة. نموذجياً، وهذا يعني تدفق التحكم وانتقاله من فعل ما إلى الذي يليه، ويتنفذ الفعل المستحوذ على التحكم حتى إتمامه، ثم يتخلى عن التحكم ويعطيه إلى الفعل الذي يليه. وسترى في الأقسام التالية أنه يمكن أن تتدفق الكائنات سوية مع التحكم عبر نشاط محدد.

٣-٢ النشاطات و الأفعال
Activities and Actions

إن الأفعال هي خطوات نشطة في إكمال عملية ما. ويمكن أن يكون العمل عبارة عن عمل حسابي، مثل "حساب الضريبة **Calculate Tax**"، أو عبارة عن مهمّة، مثل "التحقق من تفاصيل كاتب".

وتستعمل الكلمة "نشاط" في أغلب الأحيان بشكل خاطئ بدلاً من "فعل" لوصف خطوة في مخطط النشاط، لكنهما مختلفتان. إن النشاط

هو العملية المنمذجة، مثل النشاط "غسيل السيارة". والفعل هو خطوة في النشاط العام، مثل الأفعال صَوبنة، شطف، وتجفيف.



شكل رقم (٣-٣) مخطط النشاط للنشاط غسل السيارة الذي يضم الأفعال الثلاثة صوبنة Lather و شطف Rinse و تجفيف Dry.

يعرض الشكل رقم (٣-٣) أفعال النشاط البسيط "غسيل السيارة"، حيث تمت إحاطة كامل النشاط بمستطيل مدوّر الزوايا يسمى إطار النشاط **activity frame**. ويستعمل إطار النشاط للإحاطة بأفعال النشاط، وهو مفيد لإظهار أكثر من نشاط واحد على نفس المخطط. ويكتب اسم النشاط في الزاوية العليا عن يسار إطار النشاط.

إن استعمال إطار النشاط أمر اختياري، وغالباً ما يتم حذفه من مخطط النشاط، كما هو معروض في النشاط البديل لـ "غسيل سيارة" في الشكل رقم (٣-٤).

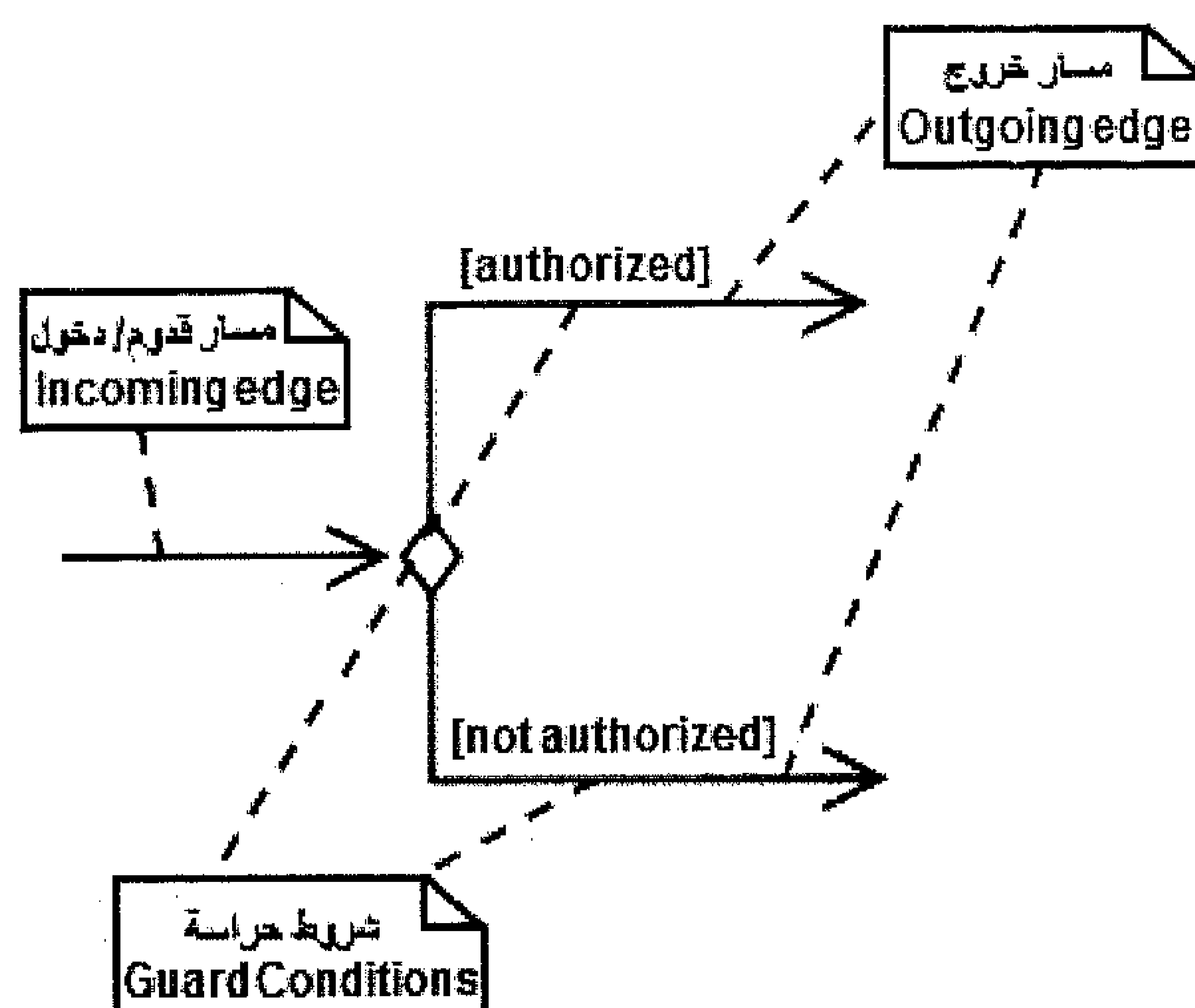


شكل رقم (٣-٤) يمكن حذف إطار النشاط.

بالرغم من فقدان اسم النشاط كونه معروض على المخطط نفسه، فغالباً ما يناسب أكثر إهمال إطار النشاط عند بناء مخطط نشاط بسيط.

٣-٣ القرارات و الاندماجات
Decisions and Merges

تستعمل القرارات عندما نريد تنفيذ سلسلة مختلفة من الأفعال بالاعتماد على شرط ما. وترسم القرارات كعقد على شكل معيّن مع مسار دخول واحد ومسارات خروج متعدّدة، كما يظهر في الشكل رقم (٣-٥).



شكل رقم (٣-٥) يتم اتباع مسار واحد فقط بعد عقدة القرار.

يحتوي كل مسار متفرع على شرط حراسة **guard condition** يكتب بين قوس [ ]. وتحدد شروط الحراسة أي مسار سيتم إتباعه بعد عقدة القرار، وهي عبارة عن تعابير شرطية يتم تقييمها فتأخذ القيمة صح أو القيمة خطأ، على سبيل المثال:

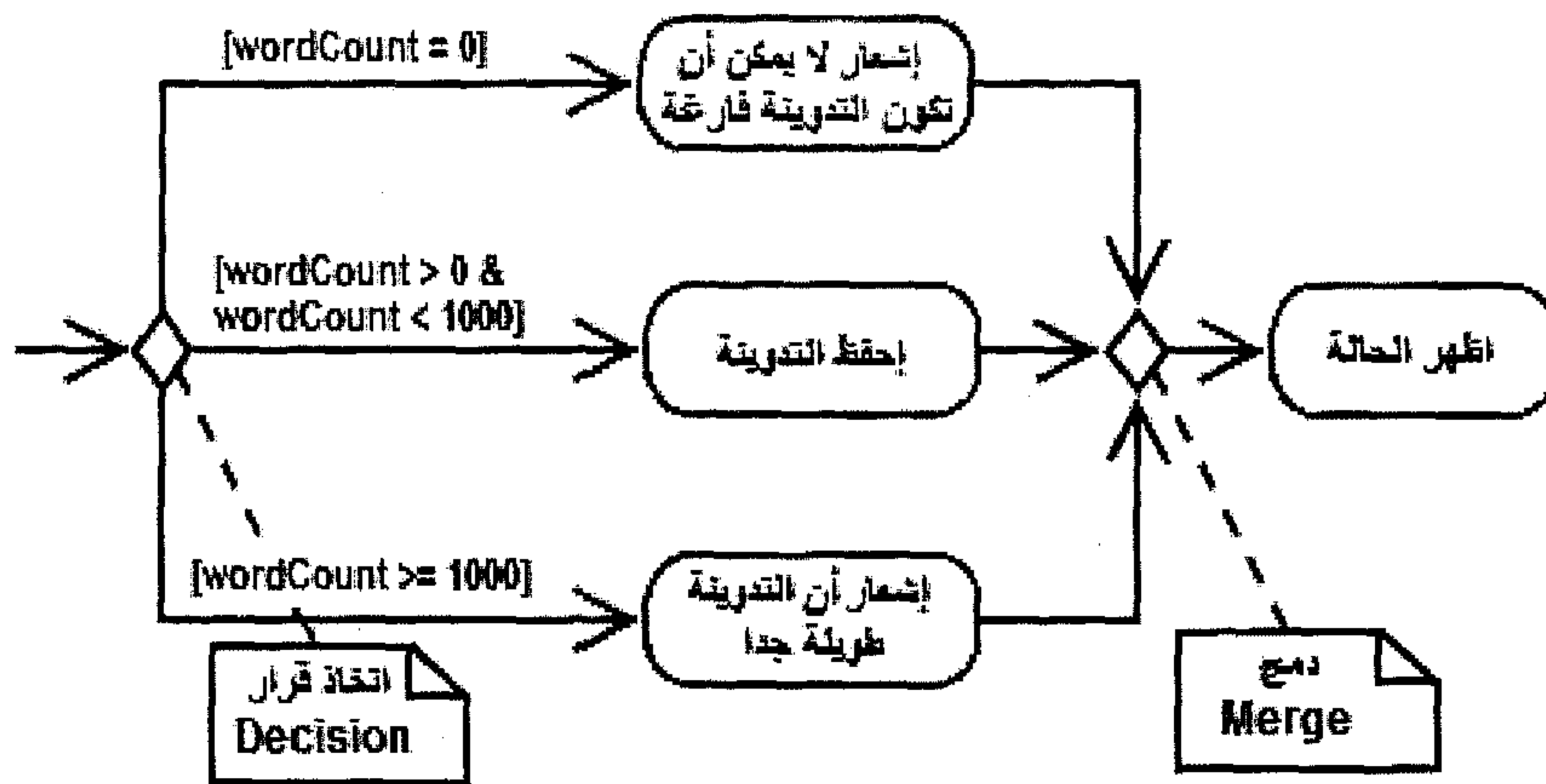
- [authorized]:

إذا تم تقييم المتغير authorized (تعني بأنه مفوّض) إلى القيمة صح، فيتم حينئذ اتباع مسار الخروج الخاص به.

- [wordCount >= 1000]:

إذا كان المتغير wordCount (عدد الكلمات) أكبر من أو يساوي ١٠٠٠، فيتم حينئذ اتباع مسار الخروج الخاص به.

تتحد التدفقات المتفرعة سوية في عقدة دمج **merge** تحدد نهاية السلوك الشرطي الذي بدأ عند عقدة القرار. وتعرض أيضاً الاندماجات كعقد لها شكل معيّن، ولكن يكون لها عدة مسارات دخول ومسار خروج واحد فقط كما هو معروض في الشكل (٣-٦).

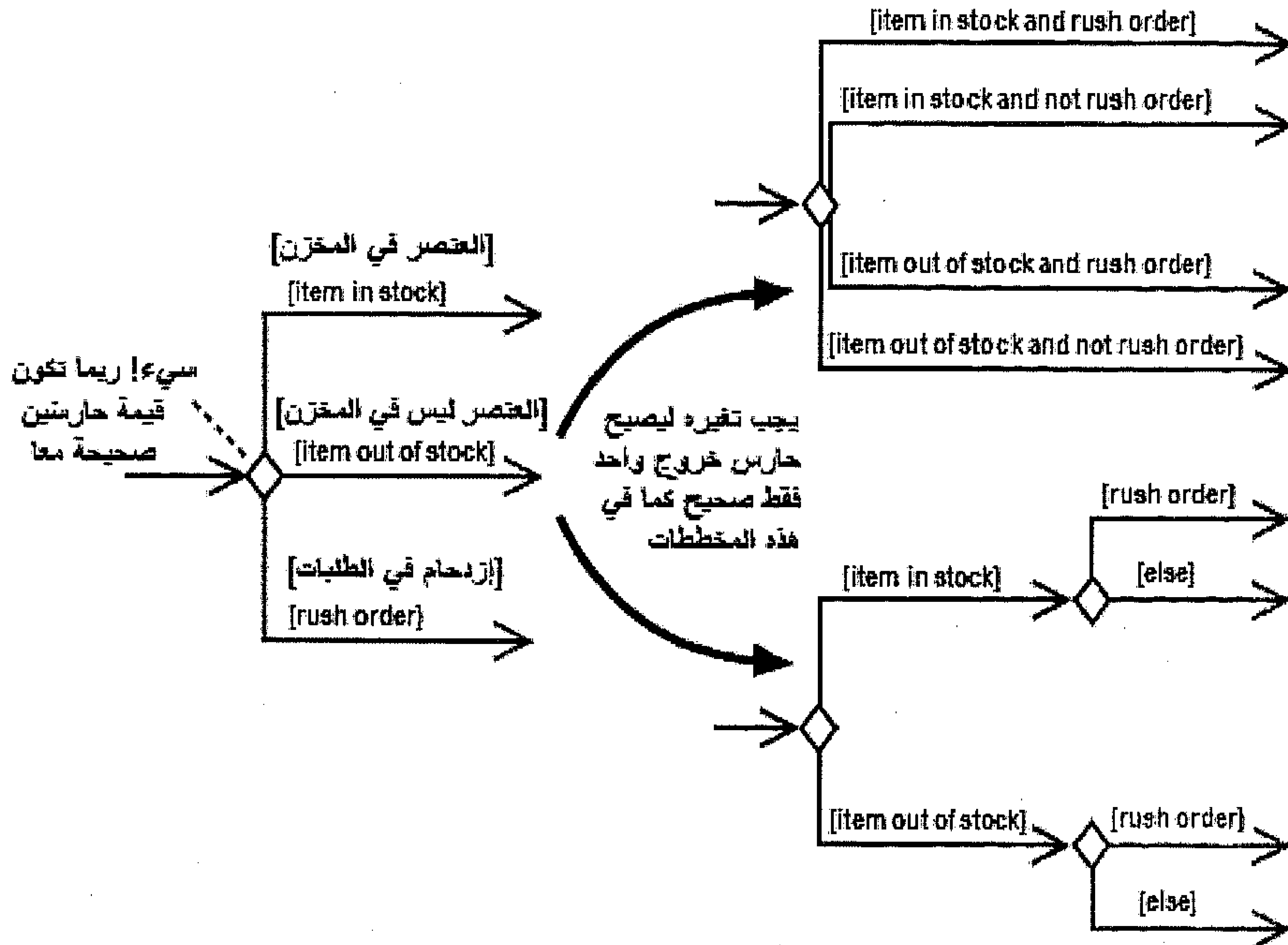


شكل رقم (٣-٦) إذا كانت قيمة المتغير **wordCount** (عدد الكلمات) أكبر من أو تساوي ١٠٠٠، فسيتم بالتالي إنجاز الفعل "إشعار أن التدوينة طويلة جدا".

تكون مخططات النشاط واضحة إذا كانت شروط الحراسة في عُقد القرار كاملة (أي تأخذ بالاعتبار جميع الحالات الممكنة) وقيمها صحيحة بالإقصاء التبادلي فيما بينها mutually exclusive (أي لا تكون

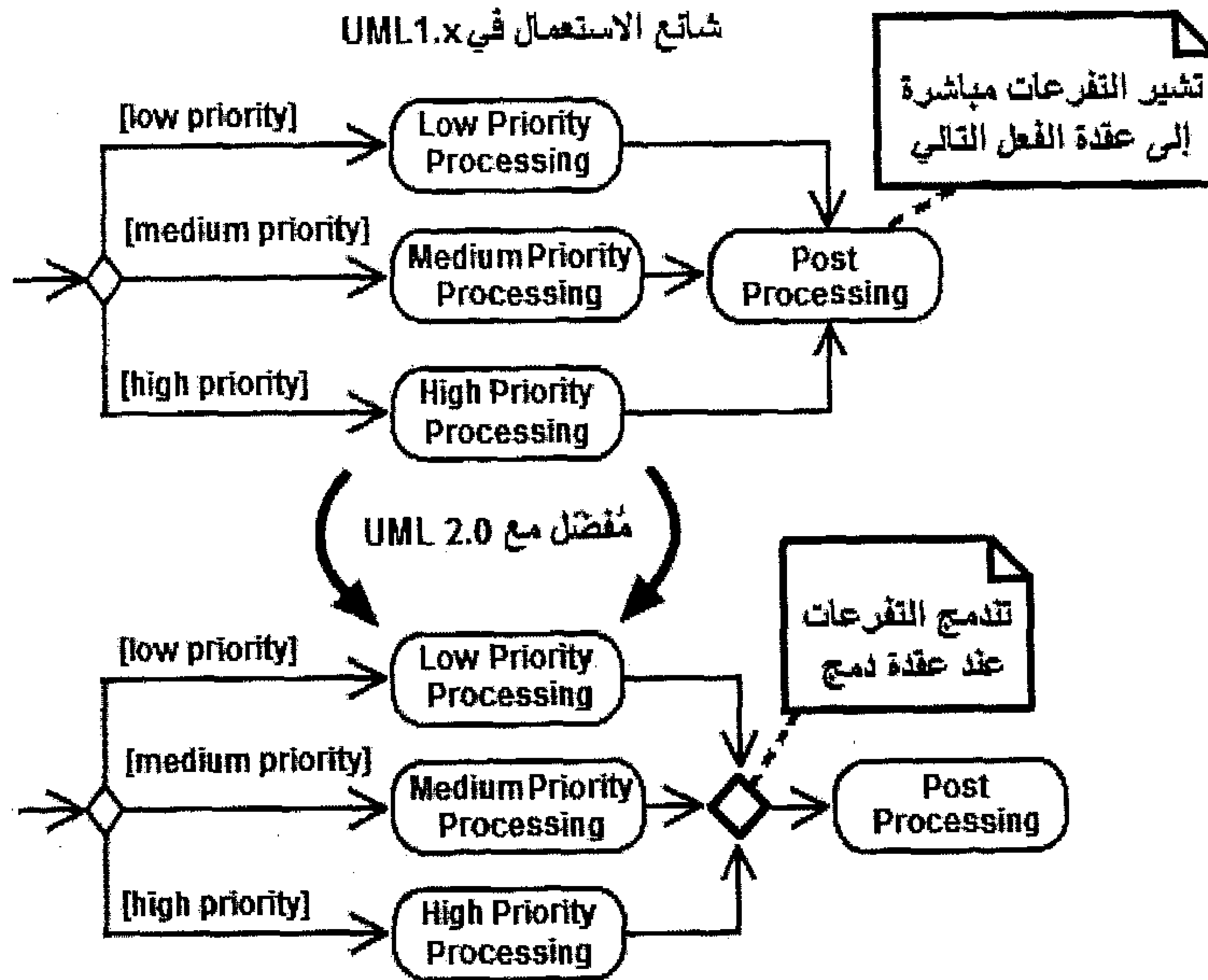
قيمة أكثر من شرط حراسة واحد فقط صح بنفس الوقت). ويعرض الشكل رقم (٣-٧) حالة لا تكون فيها المسارات صحيحة بالإقصاء التبادلي فيما بينها.

إذا تواجد عنصر في المخزن و كانت "الطلبية" مستعجلة، فيتم حينئذ تقييم شرطَي حراسة على القيمة صح؛ لذا أي مسار خاص بهذين الشرطَين سيتم اتباعه؟ وفقاً لمواصفات UML، إذا قيّمت عدة شروط حراسة على القيمة صح، فيتم حينئذ اتباع مساراً واحداً فقط، ويكون اختيار المسار خارجاً عن سيطرتك ما لم تحدّد ترتيباً معيناً لاختيار المسارات. ويمكنك تفادي هذه الحالة المعقّدة بجعل شروط الحراسة تعمل بالإقصاء التبادلي فيما بينها (أي يكون واحد منها فقط صحيحاً بوقت محدد).



شكل رقم (٣-٧) احذر المخططات المحتوية على عدة حرّاس قيمها صح بنفس الوقت.

إن الحالة الأخرى التي يجب تفاديها هي شروط الحراسة غير الكاملة. على سبيل المثال، إذا كان الشكل رقم (٣- ٧) ليس لديه شرط حراسة يغطي الحالة "العنصر ليس في المخزن"، فلا يمكن بالتالي لهذه الحالة من اتباع أي مسار خروج من عقدة القرار. وهذا يعني بقاء النشاط مجمّداً عند عقدة القرار وعدم تمكنه من المتابعة. يتخلى المنمذجون أحياناً عن شروط الحراسة إذا توقعوا عدم حدوث حالة ما (أو أرادوا إرجاء التفكير في الموضوع إلى وقت لاحق)، ولكن لتقليل الالتباس، يجب دائماً إدراج شروط حراسة لتغطية كل الحالات المحتملة. ومن المفيد أيضاً عنونة مسار خاص بالحالة "وإلا else" إذا كان ذلك مناسباً، وذلك للتأكد من تغطية كل الحالات، كما هو معروض في الشكل رقم (٣-٧).



شكل رقم (٣-٨) في UML 2.0، يفضل أن نكون واضحين قدر الإمكان بإظهار عقد الدمج.

إذا كـان لـديك خلفيـة خلفيـة عـن UML 1.x ، لـيس مـن الضـروري استعراض عقد الدمج. فمن الشائع في UML 1.x رؤية عدة مسارات تبدأ عند عقدة قرار وتنتقل مباشرة إلى فعل ما، كما هو معروض في الجزء الأعلى من الشكل رقم (٣-٨). وهذا يعنى أنه تم دمج التدفقات أو المسارات ضمنياً.

بدءاً من UML 2.0 ، عندما تعود عدة مسارات مباشرة إلى فعل ما، فتتم خدمة وإنهاء كل التدفقات القادمة قبل مواصلة التقدم. ولكن ليس لهذا معنى بسبب اتباع مسار خروج واحد فقط من عقدة القرار. ويمكن تجنّب إرباك القارئ بإظهار عُقد الدمج بشكل صريح.

٣-٤ القيام بعدة مهام في نفس الوقت
Doing Multiple Tasks at the Same Time

لندرس مسألة تدفق عمل workflow "تجميع الحاسب" الذي ينطوي على الخطوات التالية:

١- جهّز الصندوق الرئيسي case.
٢- جهّز اللوحة الأساسية motherboard.
٣- نصّب اللوحة الأساسية.
٤- نصّب السواقات drives.
٥- نصّب بطاقة الفيديو و بطاقة الصوت و المودم.

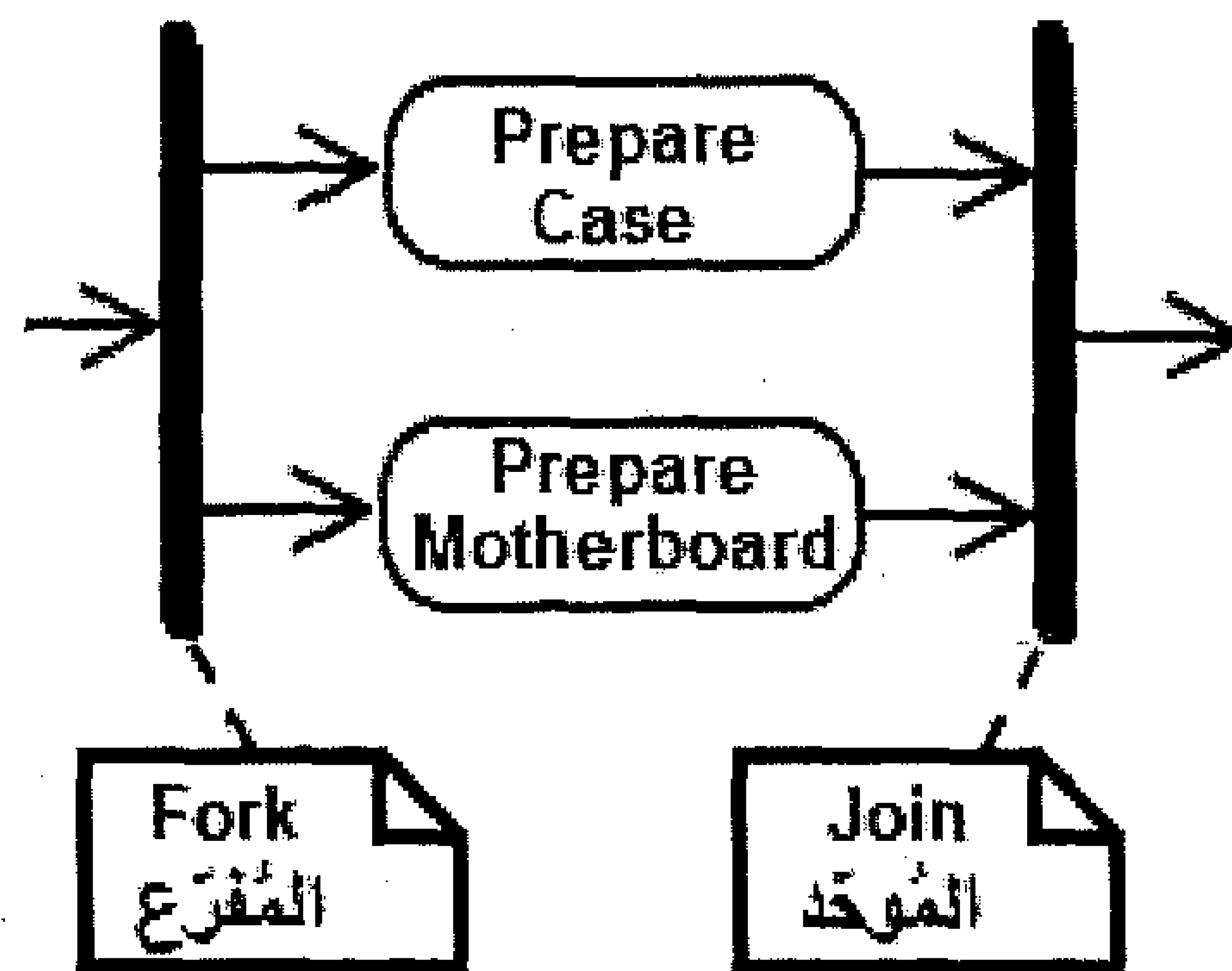
لقد رأينا حتى الآن قدراً كافياً من ترميزات مخطط النشاط لنمذجة هذا التدفق للعمل بشكل تتابعي. لكن افترض أنه يمكن تسريع تدفق العمل من خلال تجهيز الصندوق واللوحة الأساسية في نفس الوقت، لأن هذه الأفعال لا تعتمد على بعضها البعض. ويقال للخطوات

التي تحدث في نفس الوقت أنها تحدث بالتنافس **concurrently** أو بالتوازي **parallel**.

يتم تمثيل الأفعال المتوازية في مخططات النشاط باستعمال المُفرّعات **forks** والمُوحّدات **joins**، كما هو معروض في جزء مخطط النشاط في الشكل رقم (٣-٩).

بعد التفريع في الشكل رقم (٣-٩)، يتفرّع التدفق إلى تدفقين متزامنين أو أكثر، ويتم تنفيذ الأفعال الموازية لكل التدفقات الخارجة من المفرّع. وفي الشكل رقم (٣-٩) يتم تنفيذ الفعل "جهّز الصندوق **Prepare the Case**"، والفعل "حضّر لوحة أساسية **Prepare Motherboard**" في نفس الوقت بشكل متواز.

ويعني الموحّد join أنه يجب إنهاء كل الأفعال الداخلة إليه قبل التمكن من مواصلة التدفق بعده. وتبدو المفرّعات والموحّدات متطابقة - تُرسم كلتاهما باستعمال شريط عريض - لكن يمكنك تمييز الاختلاف بينهما؛ لأن المفرّعات لها تدفقات خارجة متعددة، بينما الموحّدات لها تدفقات داخلة متعددة.



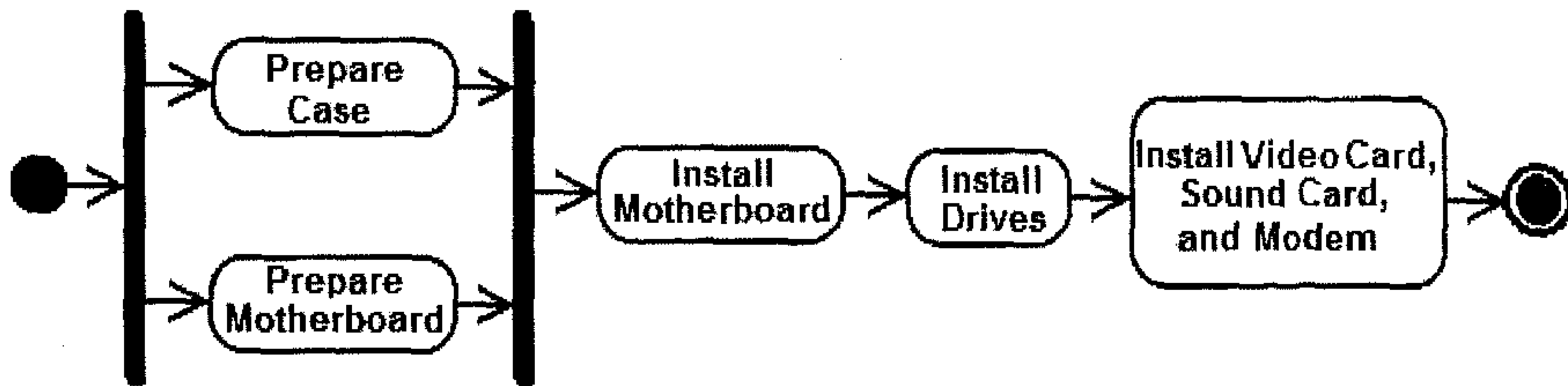
شكل رقم (٣-٩) يتم اتباع كلا مساري الخروج من المُفرّع، بالمقارنة مع عُقد القرار حيث يتم إتباع مسار خروج واحد فقط منها.



مع نموذج التصميم المفصل، يمكن استعمال المفرّعات لتمثيل العمليات المتعددة أو المسالك المتعددة **multiple threads** في برنامج ما.

يُكمل الشكل رقم (٣-١٠) مخطط النشاط لتدفق عمل تجميع الحاسب.

وعند حدوث الأفعال بشكل متواز، فهذا لا يعني بالضرورة أنها ستنتهي بنفس الوقت. وفي الحقيقة، غالباً ما تنتهي مهمة ما قبل الأخرى. وعلى أية حال، يمنع الموحّد التدفق من متابعة ما بعده حتى تصبح كل التدفقات الداخلة إليه مكتملة. وعلى سبيل المثال، في الشكل رقم (٣-١٠) ينفذ الفعل "نصّب اللوحة الأساسية **Install Motherboard**" الموجود مباشرة بعد الموحّد فقط بعد انتهاء كلا الفعلَين "جهّز صندوق **Prepare Case**" و "جهّز لوحة أساسية **Prepare Motherboard**".



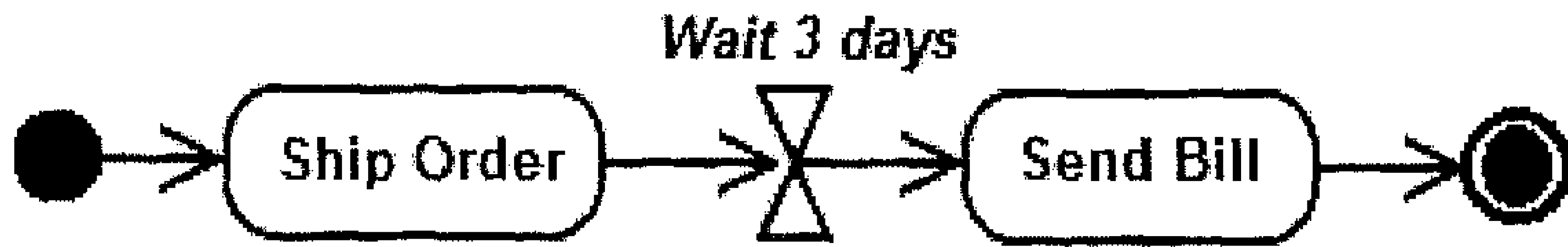
شكل رقم (٣-١٠) يبين تدفق عمل "تجميع الحاسب" عمل التفريعات و الموحّدات في مخطط نشاط كامل.

٣-٥ الأحداث الزمنية Time Events

يشكل الزمن أحياناً عاملاً في النشاط. وقد ترغب بنمذجة فترة انتظار، مثل انتظار ثلاثة أيام بعد شحن "الطلبية" لإرسال الفاتورة. وقد

تحتاج أيضاً إلى نمذجة العمليات التي تبدأ في فترات زمنية منتظمة، مثل إجراء نُسَخ احتياطية للنظام أسبوعياً.

ترسم الأحداث الزمنية باستعمال رمز الساعة الرملية. يعرض الشكل رقم (٣-١١) كيفية استعمال حدث زمني لنمذجة فترة انتظار. يبين النص - "انتظر ٣ أيام **Wait 3 Days**" - الذي بجوار الساعة الرملية زمن الانتظار. ويعني المسار الداخل إلى الحدث الزمني أنه يتم تنشيط الحدث الزمني مرّة واحدة فقط. وفي الشكل رقم (٣-١١)، يتم إرسال الفاتورة مرّة واحدة فقط وليس كل ثلاثة أيام.



شكل رقم (٣-١١) يمثل الحدث الزمني ذو المسار الداخل وقتاً مستقطعاً.

يكون الحدث الزمني الذي من دون تدفقات دخول عبارة عن حدث زمني دوري **recurring**، يعني ذلك أنه يتم تنشيطة دورياً وفقاً للتردد المذكور في النص المجاور للساعة الرملية. وفي الشكل رقم (٣-١٢)، يتم تحديث شريط التقدّم كل ثانية واحدة.



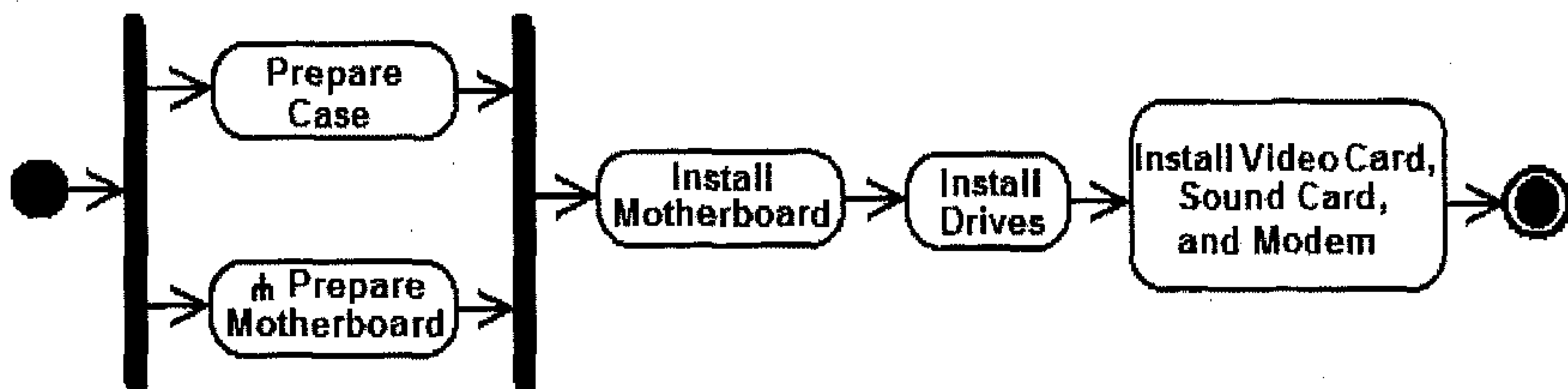
شكل رقم (٣-١٢) يمثل الحدث الزمني من دون تدفقات دخول حدثاً زمنياً دورياً.

لاحظ عدم وجود عقدة بداية في الشكل رقم (٣-١٢)؛ حيث يشكل الحدث الزمني وسيلة بديلة لبدء نشاط ما. استعمل هذا الترميز لنمذجة نشاط يتم إطلاقه بشكل دوري.

٣-٦ استدعاء نشاطات أخرى
Calling Other Activities

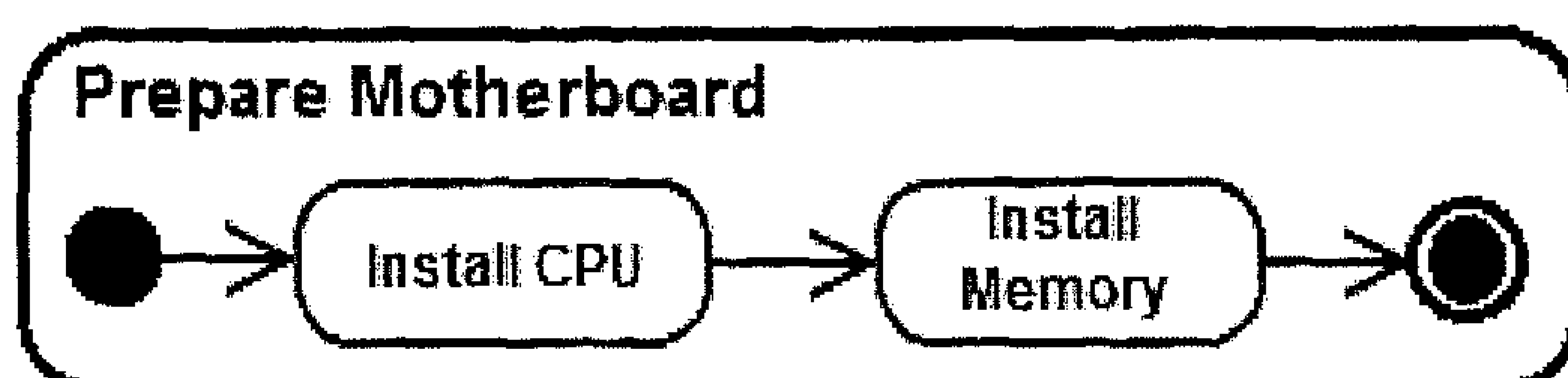
بما أنه يتم إضافة التفاصيل إلى مخطط النشاط، فقد يصبح المخطط كبيراً جداً، أو قد تحدث نفس سلسلة الأعمال أكثر من مرّة. عندما يحدث ذلك، فيمكنك تحسين مقروئية المخطط بالتزويد بتفاصيل عمل ما في مخطط منفصل، والذي يسمح لمخطط المستوى الأعلى بالبقاء بشكل أقل فوضوية.

ويعرض الشكل رقم (٣-١٣) تدفق عمل "تجميع الحاسب" من الشكل رقم (٣-١٠)، ولكن للفعل "جهّز اللوحة الأساسية **Prepare Motherboard**" الآن رمز شوكة ذراية موجهة نحو الأسفل (Ψ) تُشير إلى أنها عقدة استدعاء نشاط. وتقوم عقدة استدعاء نشاط باستدعاء النشاط المطابق لاسمها. ويشبه هذا عملية استدعاء إجراء في البرمجة.



شكل رقم (٣-١٣) بدلاً من إدخال فوضى تفاصيل الفعل "جهّز اللوحة الأساسية" على مخطط المستوى الأعلى، يتم تزويد تفاصيل هذا الفعل في مخطط نشاط آخر.

تستدعي العقدة "جهّز اللوحة الأساسية" في الشكل رقم (٣-١٣) النشاط "جهّز اللوحة الأساسية" في الشكل رقم (٣-١٤). ويتم إرفاق عقدة استدعاء النشاط بالنشاط الذي تستدعيه مع إعطائها نفس اسم العقدة. يقوم استدعاء النشاطات بتجزئة الفعل بشكل أساسي إلى أفعال مفصلة أكثر من دون أن يكون علينا إظهار كل شيء في مخطط واحد.



شكل رقم (٣-١٤) يقوم النشاط "جهّز اللوحة الأساسية" بتفصيل عملية تجهيز اللوحة الأساسية.

لمخطط النشاط "جهّز اللوحة الأساسية" عقدتا بداية ونهاية نشاط خاصتان به. تحدد عقدة النهاية نهاية النشاط "جهّز اللوحة الأساسية"، لكن هذا لا يعني اكتمال النشاط الذي قام باستدعائه. وعندما ينتهي النشاط "جهّز اللوحة الأساسية" يرجع التحكم إلى النشاط الذي قام باستدعائه لمتابعة عمله بشكل طبيعي. وهذا برهان آخر لوجه الشبه بين استدعاء النشاطات و استدعاء الإجراءات البرمجية.



بالرغم من قابلية حذف إطار النشاط مع النشاطات عالية المستوى، و لكن يجب إظهاره دائماً مع النشاطات التي يتم استدعاؤها. يساعد اسم النشاط في إطار النشاط في ربط النشاطات المُستدعات مع مُستدعيها.

٣-٧ الكائنات Objects

تكوّن أحياناً بيانات الكائنات جانباً مهماً من العملية التي نحن بصدد نمذجتها. لنفرض أن شركتك قرّرت بيع نظام إدارة محتوى CMS كمنتج تجاري، وترغب بتعريف عملية للموافقة على "الطلبيات" القادمة. تحتاج كل خطوة في هذه العملية إلى معلومات حول "الطلبية"، مثل الدفعات المالية و تكلفة الصفقة. ويمكن نمذجة هذا في مخطط النشاط من خلال الكائن طلبية **Order**، الذي يحتوي على معلومات "الطلبية" التي تحتاجها تلك الخطوات. وتقدم مخططات النشاط تشكيلة وسائل لنمذجة الكائنات في العمليات.



يمكن أن تختلف الكائنات هنا عن الكائنات البرمجية. على سبيل المثال، في نشاط تجميع غير آلي لحاسب، قد تستعمل عقدة كائن لتمثيل طلب عمل مادي يقوم ببدء العملية.

٣-٧-١ إظهار الكائنات الممررة بين الأفعال

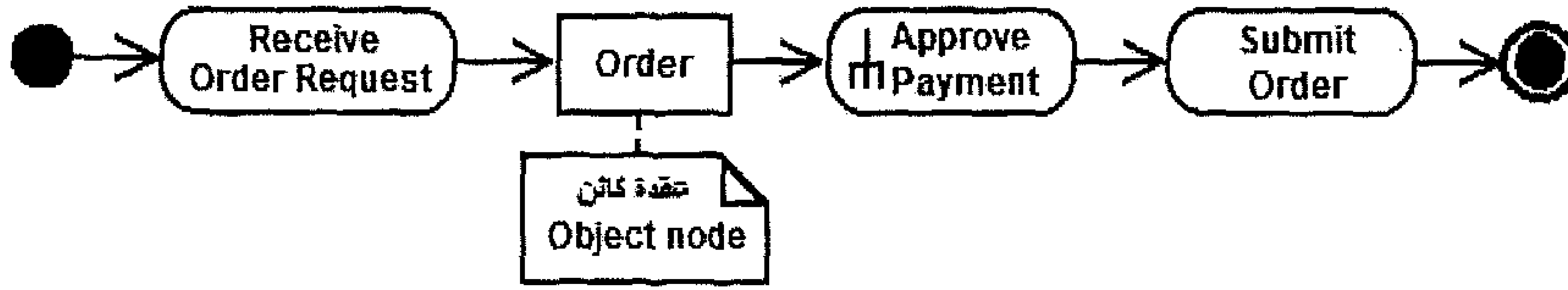
Showing Objects Passed Between Actions

في مخططات النشاط، يمكن استعمال عُقد كائن لإظهار تدفق البيانات عبر النشاط. وتسمح عقدة الكائن بتمثيل كائن متوفّر عند نقطة معيّنة في النشاط، ويمكن استعمالها لإظهار أن الكائن مُستعمل أو مُنشأ أو مُعدّل من قِبل أيّ من الأفعال المحيطة به.

وترسم عقدة الكائن باستعمال شكل مستطيل عادي، كما هو معروض في عملية الموافقة على الطلبية في الشكل رقم (٣-١٥). وتلفت عقدة الكائن طلبية **Order** الانتباه إلى أن الكائن طلبية يتدفق من الفعل

"استلام طلب طلبية Receive Order Request" إلى الفعل "الموافقة على الدفعة **Approve Payment**".

انظر إلى القسم "إرسال واستلام الإشارات" لتتعرف على وسيلة أكثر دقة لنمذجة الفعل "استلام طلب طلبية **Receive Order Request**" كعقدة استلام إشارة.

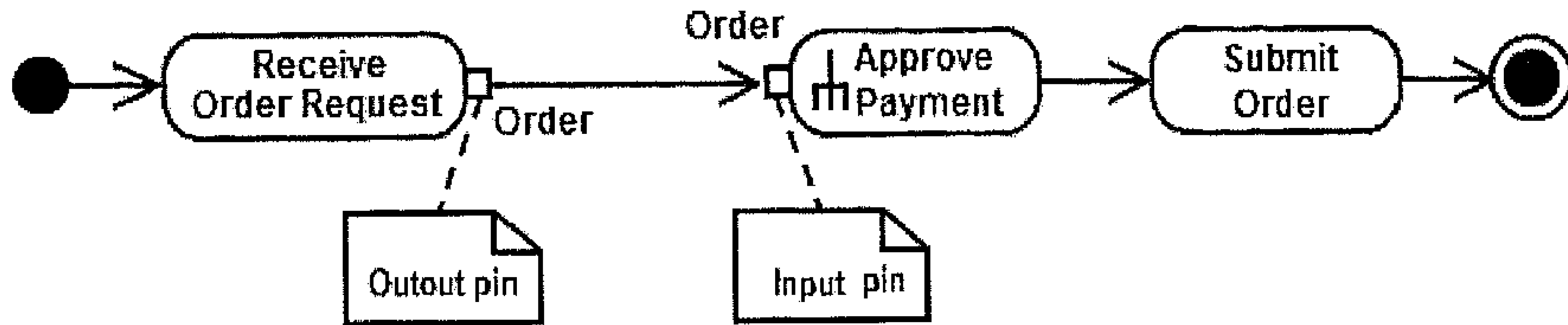


شكل رقم (٣-١٥) تركز عقدة الكائن طلبية على أهمية هذه البيانات في النشاط و تظهر الأفعال التي تتفاعل معها.

٣-٧-٢ عرض مدخلات و مخرجات الفعل
Showing Action Inputs and Outputs

يعرض الشكل رقم (٣-١٦) منظوراً مختلفاً عن النشاط السابق باستعمال الدبابيس **pins**. وتبين الدبابيس أنّ الكائن عبارة عن مُدخل إلى أو مُخرج من فعل ما.

ويعني دبوس الإدخال **input pin** أن الكائن المحدد هو مُدخل للفعل. ويعني دبوس الإخراج **output pin** أن الكائن المحدد هو مُخرج من الفعل. وفي الشكل رقم (٣-١٦)، عندنا كائن طلبية مُدخل للفعل "الموافقة على الدفعة" وكائن طلبية مُخرج من الفعل "استلام طلب طلبية".

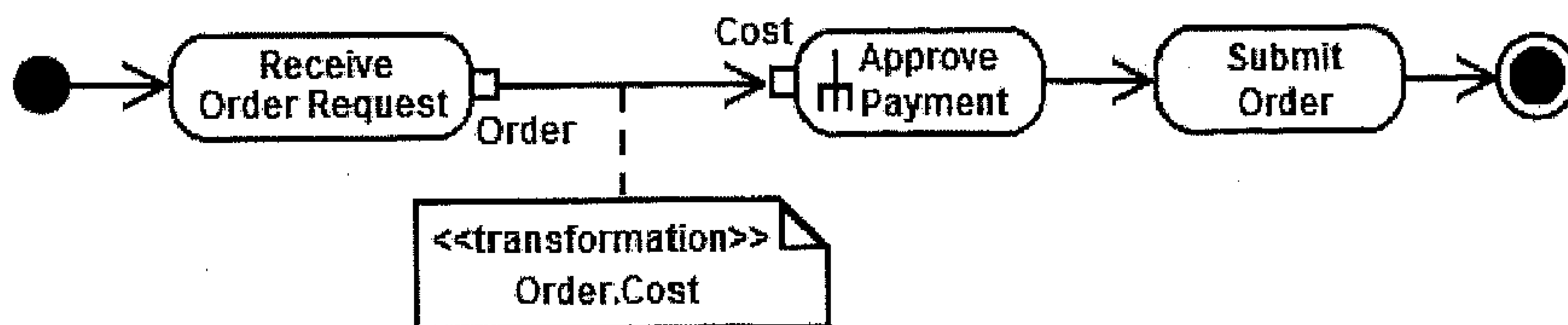


شكل رقم (٣-١٦) تتطلب الدبابيس في هذا التغيير بعملية موافقة تسمح بتوصيف منفصل ودقيق لبارامترات الإدخال والإخراج.

يظهر الشكلان رقم (٣-١٥) ورقم (٣-١٦) حالات متماثلة، لكن تكون الدبابيس جيدة للتأكيد بأن الكائن هو مُدخل أو مُخرج مطلوب، بينما تعني عقدة الكائن ببساطة أن الكائن متوفّر عند النقطة المحددة في النشاط. وهي جيدة للتأكيد على تدفق البيانات عبر النشاط.

إذا احتاج الفعل "الموافقة على الدفعة" إلى أجزاء فقط من كائن طلبية - ليس إلى كامل الكائن - يمكن استعمال تحويل لإظهار أي الأجزاء ضرورية. وتسمح التحويلات بإظهار كيف أن المُخرج من فعل ما يوفر المُدخل إلى فعل آخر.

يشير الشكل رقم (٣-١٧) أنّ الفعل "الموافقة على الدفعة" يتطلب كائن كلفة **Cost** كمُدخل له، ويظهر كيفية الحصول على هذه البيانات من كائن طلبية **Order** باستعمال عملية التحويل المحددة في ملاحظة.



شكل رقم (٣-١٧) تظهر التحويلات من أين تأتي بارامترات المدخلات.

٣-٧-٣ إظهار كيفية تغيير الكائنات لحالتها أثناء النشاط

Showing How Objects Change State During an Activity

يمكن أيضاً إظهار كائن يُغيّر حالته خلال تدفقه عبر النشاط. يظهر الشكل رقم (٣-١٨) أن حالة كائن طلبية **Order** تكون معلّقة **pending** قبل الفعل "الموافقة على الدفعة" ثم تتغير بعدئذ إلى الحالة "موافق عليها **approved**". ويتم إظهار الحالة بين قوسين [].



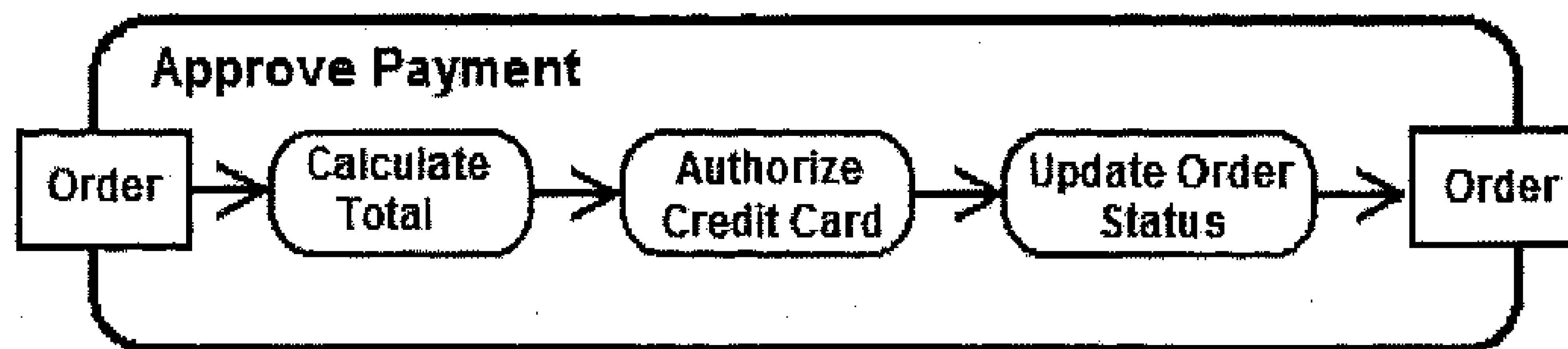
شكل رقم (٣-١٨) يركز هذا المخطط على تغيير حالة كائن طلبية خلال مراحل عملية الموافقة على طلبية.

٣-٧-٤ إظهار مُدخل و مُخرج النشاط

Showing Input to and Output from an Activity

بالإضافة إلى عملها كمدخلات ومخرجات للأفعال، فيمكن أن تكون عُقد الكائنات مُدخلات ومُخرجات للنشاطات. وترسم مُدخلات ومُخرجات النشاط كعُقد عن جانبي إطار النشاط، كما هو معروض في الشكل رقم (٣-١٩). ويفيد هذا الترميز في التأكيد على تطلب كامل النشاط لمُدخل و توفيره لمُخرج.

ويظهر الشكل رقم (٣-١٩) كائن طلبية كمُدخل ومُخرج للنشاط "الموافقة على الدفعة". وعند ظهور بارامترات الإدخال والإخراج يتم حذف عقدتي بداية ونهاية النشاط.



شكل رقم (٣-١٩) يمكن استعمال عُقد الكائن للتأكيد على مُدخل و مُخرج للنشاط.

٣-٨ إرسال واستلام الإشارات
Sending and Receiving Signals

ربما تستلزم النشاطات التفاعل مع أشخاص أو أنظمة أو عمليات خارجية. وعلى سبيل المثال، عند السماح بالدفع ببطاقة ائتمان، وتحتاج إلى التحقق من البطاقة بالتفاعل مع خدمة المصادقة التي توفرها شركة بطاقة الائتمان.

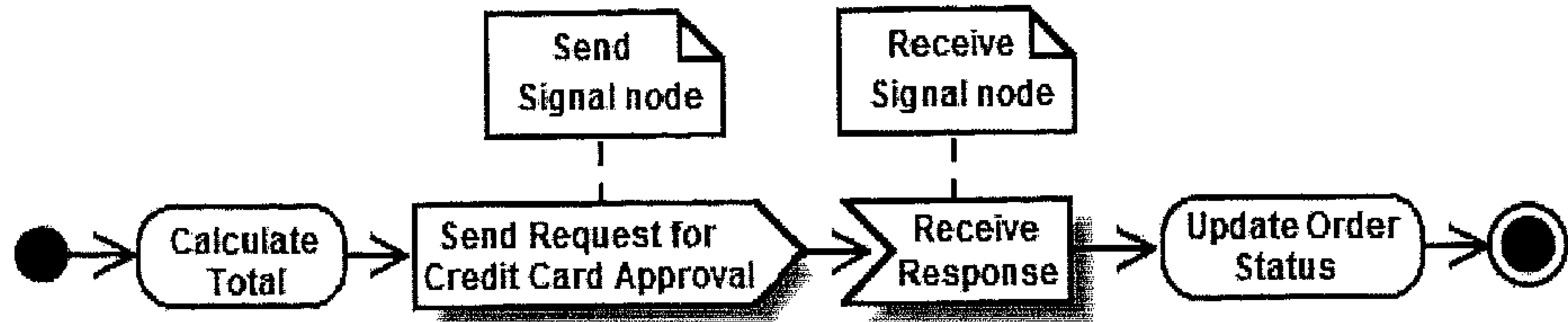
في مخططات النشاط، تمثل الإشارات **signals** التفاعلات التي تتم مع المشاركين الخارجيين. والإشارات هي الرسائل التي يمكن إرسالها أو استلامها، كما في الأمثلة التالية:

- يرسل البرنامج طلباً إلى شركة بطاقة الائتمان للمصادقة على صفقة تم إبرامها باستعمال البطاقة، ويستلم البرنامج ردّاً من الشركة (إرسال واستلام من منظور نشاط المصادقة على بطاقة الائتمان).
- يدفع استلام الطلبية إلى بدء عملية معالجة الطلبية (استلام من منظور نشاط معالجة الطلبية).
- يسبّب النقر على الزر بتنفيذ الشفرة المرافق للزر (استلام من منظور نشاط معالجة حدث الزر).
- يُعلم النظام العميل بتأخير شحنته (إرسال من منظور نشاط شحن الطلبية).

إن تأثير إشارة الاستلام **receive signal** هو إيقاظ فعل ما في مخطط نشاط. ويعرف مستلم الإشارة كيفية الاستجابة لها، ويتوقع وصول إشارة ما في وقت محدد لكن من دون معرفة متى بالضبط. إن إشارات الإرسال **send signals** هي إشارات تم إرسالها إلى مشارك خارجي. وعندما

يستلم ذلك الشخص أو النظام الخارجي الرسالة، فمن المحتمل أن يعمل شيئاً بالمقابل، ولكن هذا لا ينمذج في مخطط نشاط.

ينقّح الشكل رقم (٣-٢٠) خطوات الشكل رقم (٣-١٩) لإظهار تطلب عملية المصادقة على بطاقة الائتمان تفاعلاً مع برنامج خارجي. وتشير عقدة إرسال إشارة إلى أنه قد تم إرسال إشارة إلى مشترك خارجي. وفي هذا المثال، الإشارة هي طلب المصادقة على بطاقة الائتمان. ويتم إرسال الإشارات بشكل غير متزامن، وهذا يعني أن النشاط لا ينتظر الردّ لكن يتقدم مباشرة إلى الفعل التالي بعد إرسال الإشارة.



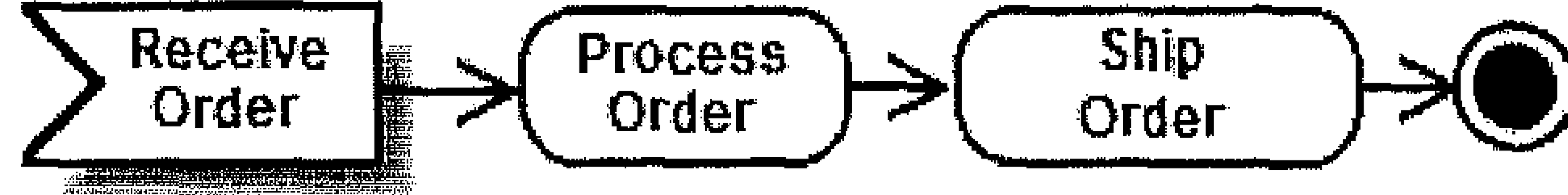
شكل رقم (٣-٢٠) تظهر عُقد إرسال واستلام إشارات التفاعلات مع المشاركين الخارجيين.

تظهر عقدة استلام إشارة أنه قد تم استلام إشارة ما من عملية خارجية. وفي هذه الحالة، انتظار النظام ردّاً من شركة بطاقة الائتمان. وعند عقدة استلام إشارة، ينتظر الفعل حتى استلام إشارة ما ويتابع فقط بعد استلام الإشارة.



> لاحظ أن المزج بين إشارات الإرسال و الاستلام ينتج سلوكاً مشابهاً لاستدعاء متزامن، أو استدعاء ينتظر ردّاً. من الشائع دمج إشارات الإرسال والاستلام في مخططات النشاط بسبب الحاجة إلى رد ما على الإشارة المرسلة.

عند وجود عقدة استلام إشارة من دون تدفقات دخول، تكون العقدة تنتظر بشكل دائم إشارة ما عندما يكون النشاط الذي يحتويها نشطاً. وفي حالة الشكل رقم (٣-٢١)، يتم بدء النشاط عند كل مرّة يتم استلام إشارة طلب حساب.



شكل رقم (٣-٢١) بدء نشاط ما بعقدة استلام إشارة: تحل عقدة استلام الإشارة مكان عقدة البداية العادية.

يختلف هذا عن عقدة استلام إشارة مع مسار دخول، مثل العقدة "استلام رد Receive Response" في الشكل رقم (٣-٢٠)؛ تبدأ عقدة استلام إشارة مع مسار دخول بالانتظار فقط عند اكتمال الفعل السابق.

٣-٩ البدء في النشاط
Starting an Activity

الطريق الأسهل والأكثر شيوعاً للبدء في نشاط محدد هو باستعمال عقدة بداية واحدة؛ وتستعمل معظم المخططات السابقة في هذا الفصل هذا الترميز. وتوجد طرق أخرى لتمثيل بداية النشاط ولكنها تحمل معان خاصّة:

- يبدأ النشاط من خلال استلام بيانات مدخلة، كما هو معروض سابقاً في القسم "إظهار مُدخل و مُخرج النشاط".
- يبدأ النشاط استجابة إلى حدث زمني، كما هو معروض سابقاً في القسم "الأحداث الزمنية".
- يبدأ النشاط كنتيجة لإيقاظه من قبل إشارة ما.

استعمل عقدة استلام إشارة بدلاً من عقدة بداية لتحديد بداية النشاط كنتيجة لإيقاظه من قبل إشارة ما. ويتم تحديد نوع الحدث الذي يبدأ النشاط داخل عقدة استلام إشارة. ويعرض الشكل رقم (٣-٢١) نشاطاً يبدأ عند استلام "طلبية".

٣-١٠ إنهاء النشاطات والتدفقات
Ending Activities and Flows

لم تكن عُقد النهاية في هذا الفصل مهمة جداً حتى الآن؛ وفي الحقيقة، هي لم تعمل أكثر من كونها محددات نهاية. وفي العالم الحقيقي، يمكن مصادفة نهايات للعمليات أكثر تعقيداً تتضمن تدفقات يمكن اعتراضها interrupted وتدفقات تنتهي من دون انتهاء النشاط العام.

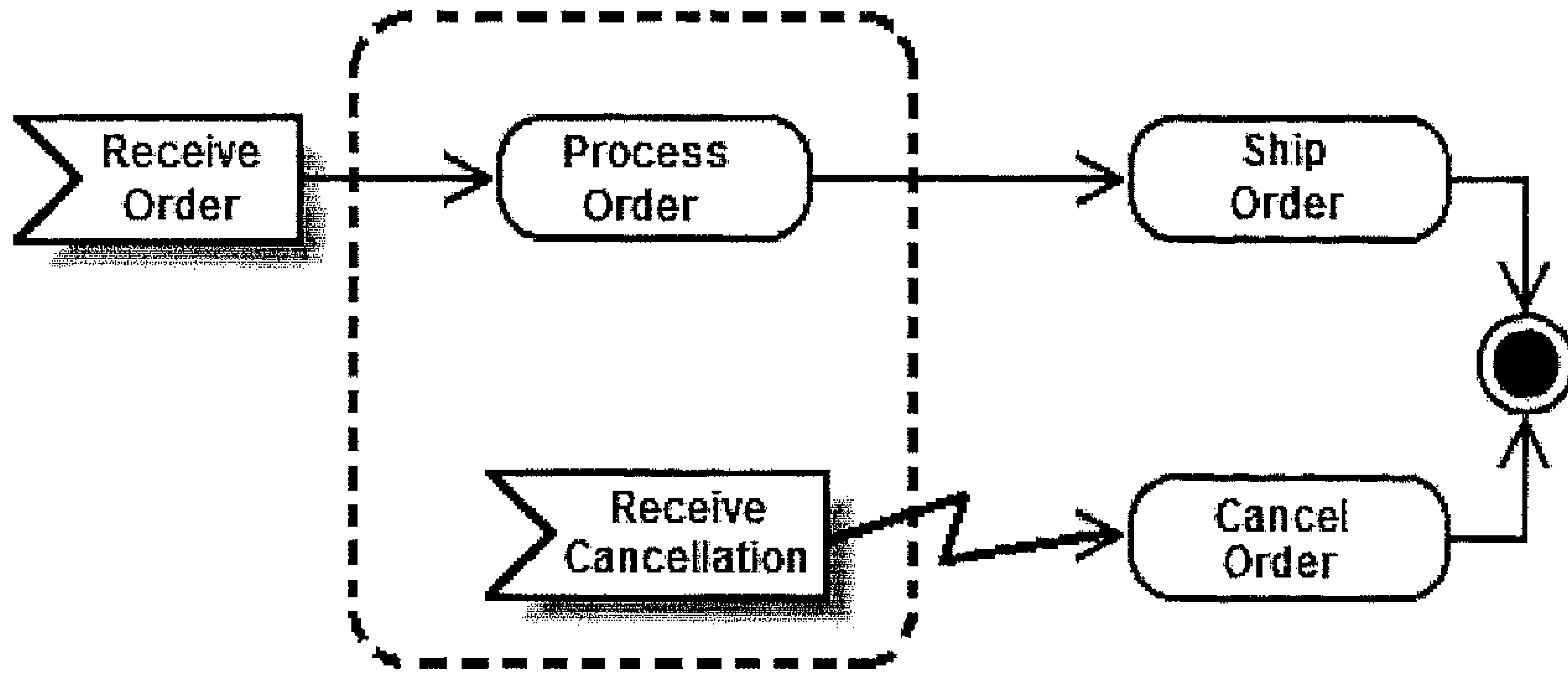
٣-١٠-١ اعتراض النشاط Interrupting an Activity

يعرض الشكل رقم (٣-٢١) السابق مخطط نشاط نموذجي مع عقدة نهاية بسيطة. لاحظ وجود مسار واحد فقط يؤدّي إلى عقدة نهاية النشاط؛ يحصل كل فعل في هذا المخطط على فرصة للانتهاء.

تحتاج أحياناً إلى نمذجة عملية قد تنتهي بحدث. يمكن حدوث ذلك عند وجود عملية طويلة التنفيذ يمكن اعتراضها من قِبَل المستخدم. أو احتمال الحاجة لتعليل إلغاء "طلبية" ضمن نشاط معالجة "طلبية" في نظام إدارة المحتوى. ويمكن إظهار الاعتراضات باستعمال مناطق الاعتراض **interruption regions**.

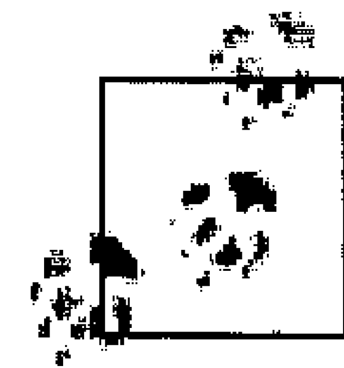
وترسم منطقة الاعتراض باستعمال مستطيل منقط مدوّر الزوايا، ويُحيط المستطيل الأفعال التي يمكن اعتراضها والحدث المتسبب

بالاعتراض. ويأتي بعد حدث الاعتراض خطاً متعرجاً يشبه البرق. ويمثل الشكل رقم (٣-٢٢) توسيعاً للشكل رقم (٣-٢١) من أجل تفسير احتمال إلغاء "الطلبية".



شكل رقم (٣-٢٢) استعمال منطقة اعتراض لإظهار إمكانية اعتراض عملية.

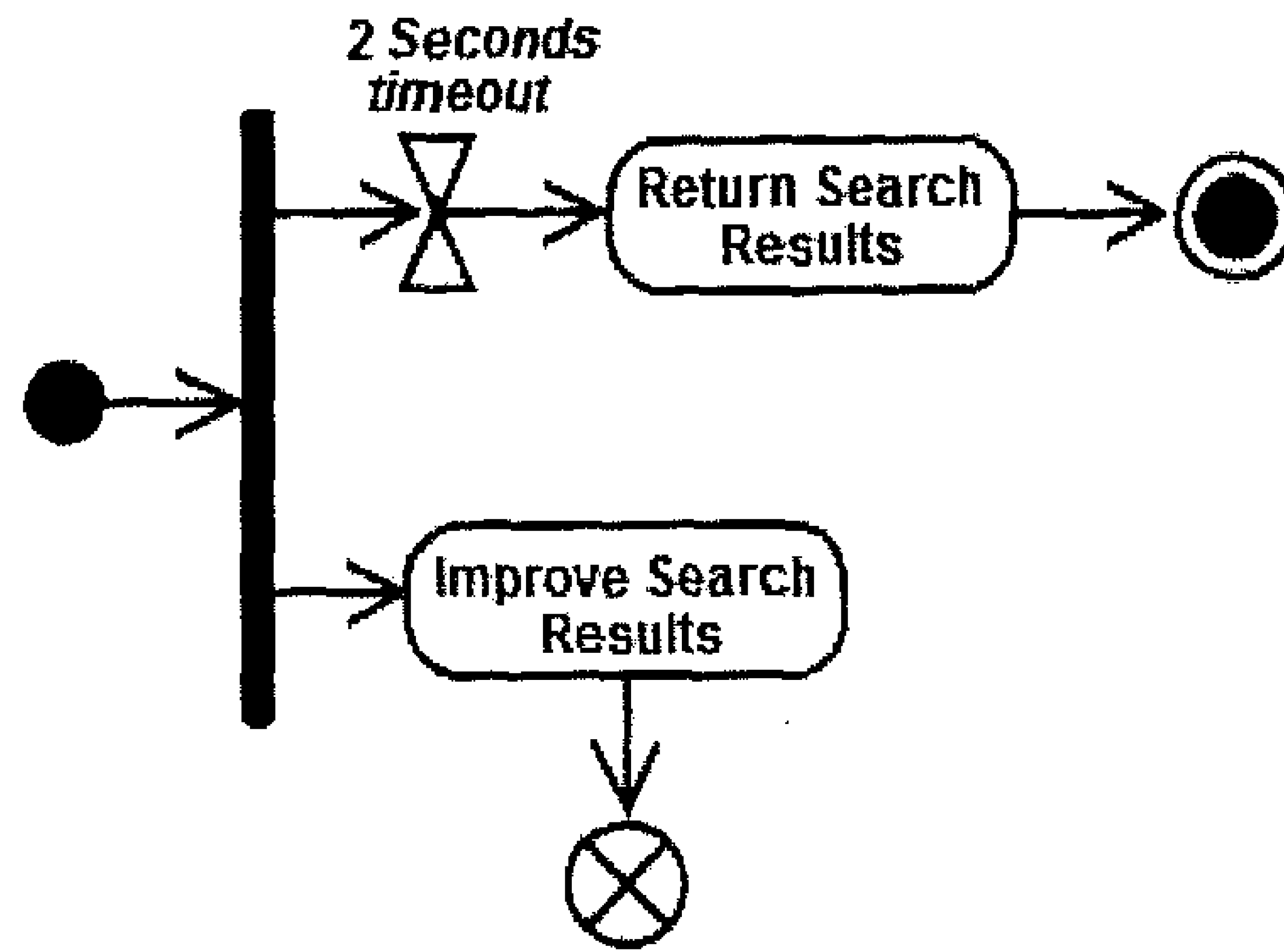
في الشكل رقم (٣-٢٢)، إذا تم استلام إلغاء طلبية خلال نشاط العملية "معالجة طلبية **Process Order**"، سيتم اعتراض هذه العملية وتنشيط العملية "إلغاء طلبية **Cancel Order**". وتكون مناطق الإلغاء مهمة بالنسبة للأفعال التي بداخلها فقط. إذا تم استلام إلغاء طلبية خلال نشاط العملية "شحن طلبية **Ship Order**"، فلن يتم اعتراض العملية "شحن طلبية" لأنها ليست داخل منطقة الإلغاء.



قد تصادف أحياناً مخططات نشاط لها عدة عُقد نهاية نشاط بدلاً من عدة تدفقات مؤدية إلى عقدة نهاية نشاط وحيدة. من الجائز إجراء ذلك و قد يُساعد في إلغاء تشابك الخطوط في مخطط كثير التفرعات. لكن تكون مخططات النشاط أسهل للفهم عند احتوائها على عقدة نهاية نشاط وحيدة.

٣-١٠-٢ إنهاء التدفق Ending a Flow

يوجد ميزة جديدة في UML 2.0 تتمثل بالتمكن من إظهار موت أو نهاية تدفقاً من دون إنهاء كامل النشاط. تقوم عقدة نهاية تدفق **flow final** بإنهاء المسار الخاص بها فقط وليس إنهاء كامل النشاط. ويتم ذلك باستعمال دائرة يتخللها الرمز X كما في الشكل رقم (٣-٢٣).



شكل رقم (٣-٢٣) ينهي تدفق المسار الخاص بعقدة النهاية فقط وليس كامل النشاط.

يعرض الشكل رقم (٣-٢٣) محرك بحث لنظام إدارة محتوى مع نافذة تظهر بعد ثانيتين لدعم توليد أفضل النتائج المحتملة للبحث. عند انقضاء ثانيتي إيقاف البحث، يتم إرجاع نتائج البحث، و ينتهي كامل النشاط بما فيه الفعل "تحسين نتائج البحث **Improve Search Results**".

على أية حال، إذا انتهى الفعل "تحسين نتائج البحث" قبل ثانيتي الخروج، لن يتوقف النشاط العام لأن تدفقه ينتهي بعقدة نهاية تدفق.



يجب الحذر عند استعمال عقدة نهاية تدفق بعد التفريع. بمجرد الوصول إلى عقدة نهاية نشاط، فتنتهي كل الأفعال الأخرى في النشاط (بما فيها العُقد التي قبل العقدة الأخيرة). إذا أردت تنفيذ كل الأفعال المتشعّبة حتى النهاية، فتأكّد من إضافة مُوحّد.

٣-١١ التجزئة Partitions
(أو ممرات السباحة Swimlanes)

يمكن مشاركة عدة مشاركين مختلفين في النشاطات، مثل المجموعات أو الوظائف المختلفة في المنظمة أو النظام. وتتطلب السيناريوهات التالية عدة مشاركين لإكمال النشاط:

١- نشاط معالجة طلب An order processing activity

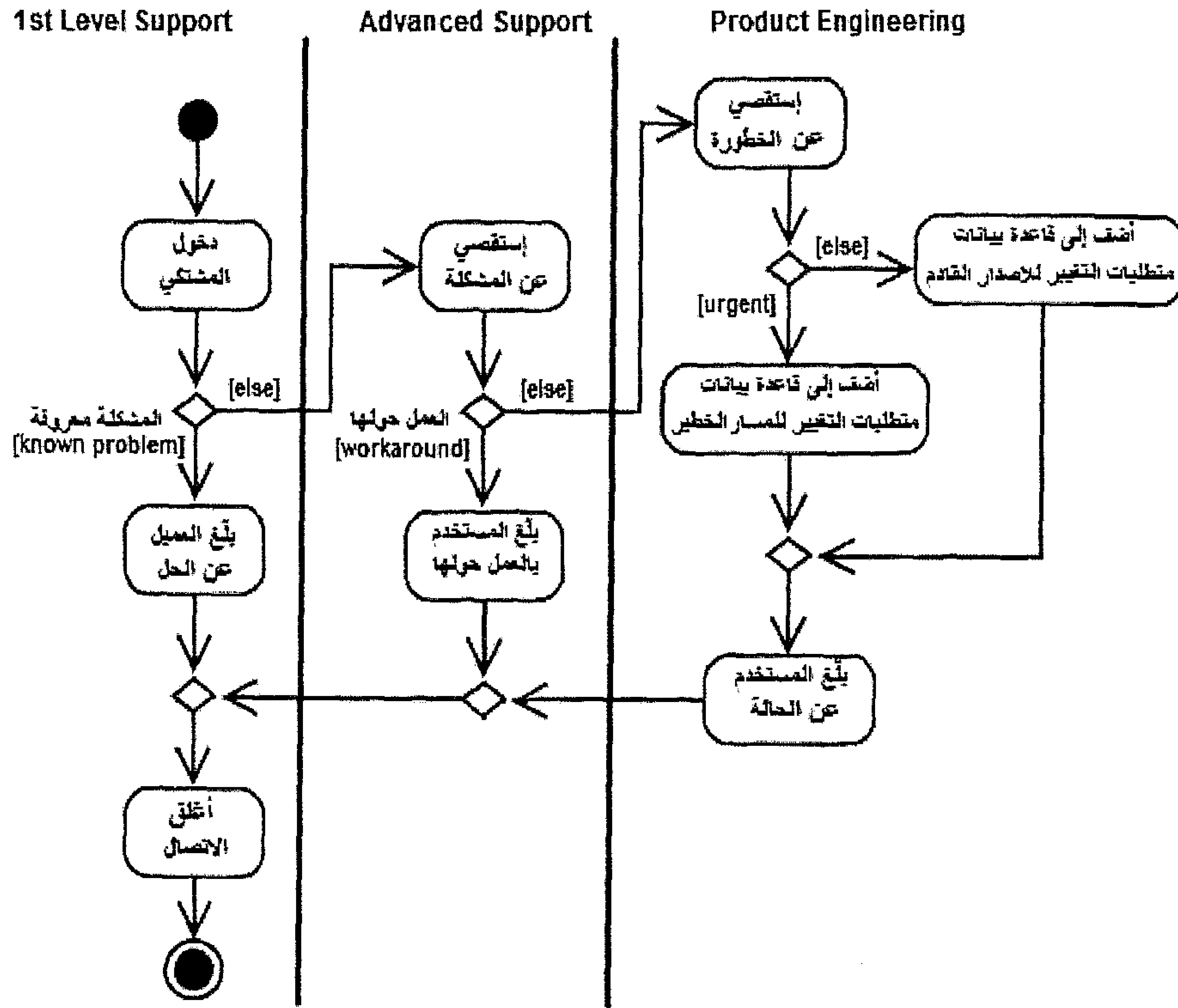
يتطلب قسم الشحن أن تشحن المنتجات ويتطلب قسم المحاسبة أن ترسل الفاتورة إلى العميل.

٢- عملية دعم فني A technical support process

تتطلب مستويات مختلفة من الدعم بما فيها دعم بمستوي أولي ودعم متقدّم وهندسة منتج.

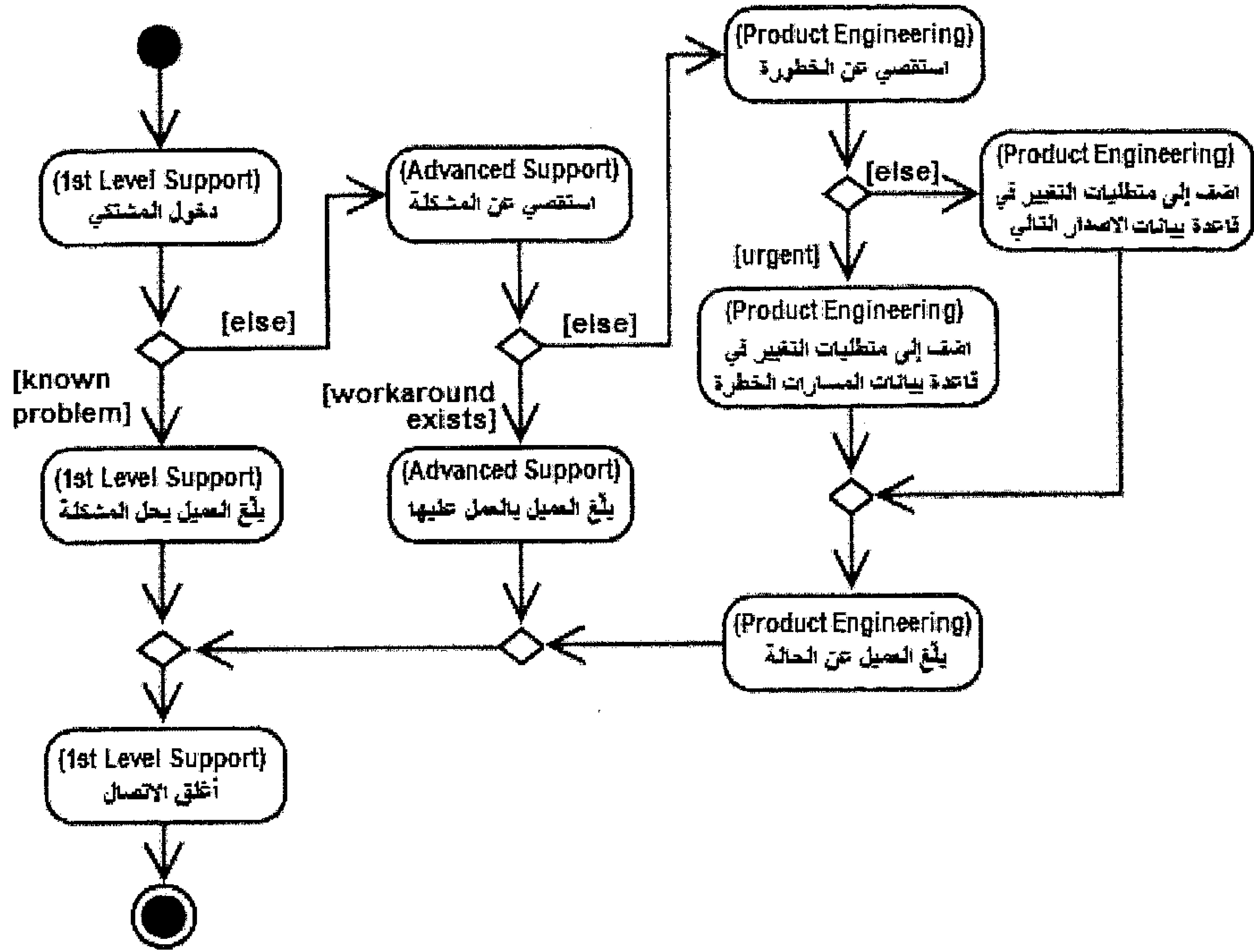
يتم استعمال التجزئة لإظهار الأفعال التي يكون مسئولاً عنها كل مشارك. وتقوم التجزئة بتقسيم المخطط إلى أعمدة أو صفوف (بالاعتماد على اتجاه مخطط النشاط)، وتحتوي على الأفعال التي تنفذ من قِبل مسئول المجموعة. ويشار أحياناً إلى الأعمدة أو الصفوف بممرات السباحة.

ويعرض الشكل رقم (٣-٢٤) عملية دعم فني يشارك فيها ثلاثة أنواع من المشاركين: دعم بمستوي أولي 1st level Support، دعم متقدّم Advanced Support، وهندسة منتج Product Engineering.



شكل رقم (٣-٢٤) تساعد التجزئة على تنظيم مخطط النشاط بتوضيح مسئولي الأجزاء.

يمكن أيضاً إظهار المسؤولية باستعمال الملاحظات داخل العقد **annotations**. لاحظ عدم وجود تجزئة تتخلل المخطط (أعمدة)؛ بدلاً من ذلك، ويوضع اسم مسئول الجزء بين قوسين () داخل العقدة، كما هو معروض في الشكل رقم (٣-٢٥). وعادة ما يجعل هذا الترميز المخطط أكثر إيجازاً، لكنه لا يظهر المشاركين بشكل واضح كالتجزئة.



شكل رقم (٣-٢٥) يمكن استعمال الملاحظات داخل العقد بدلاً من التجزئة كوسيلة لإظهار المسؤولية مباشرة داخل الفعل.

٣-١٢ إدارة مخططات نشاط معقدة
Managing Complex Activity Diagrams

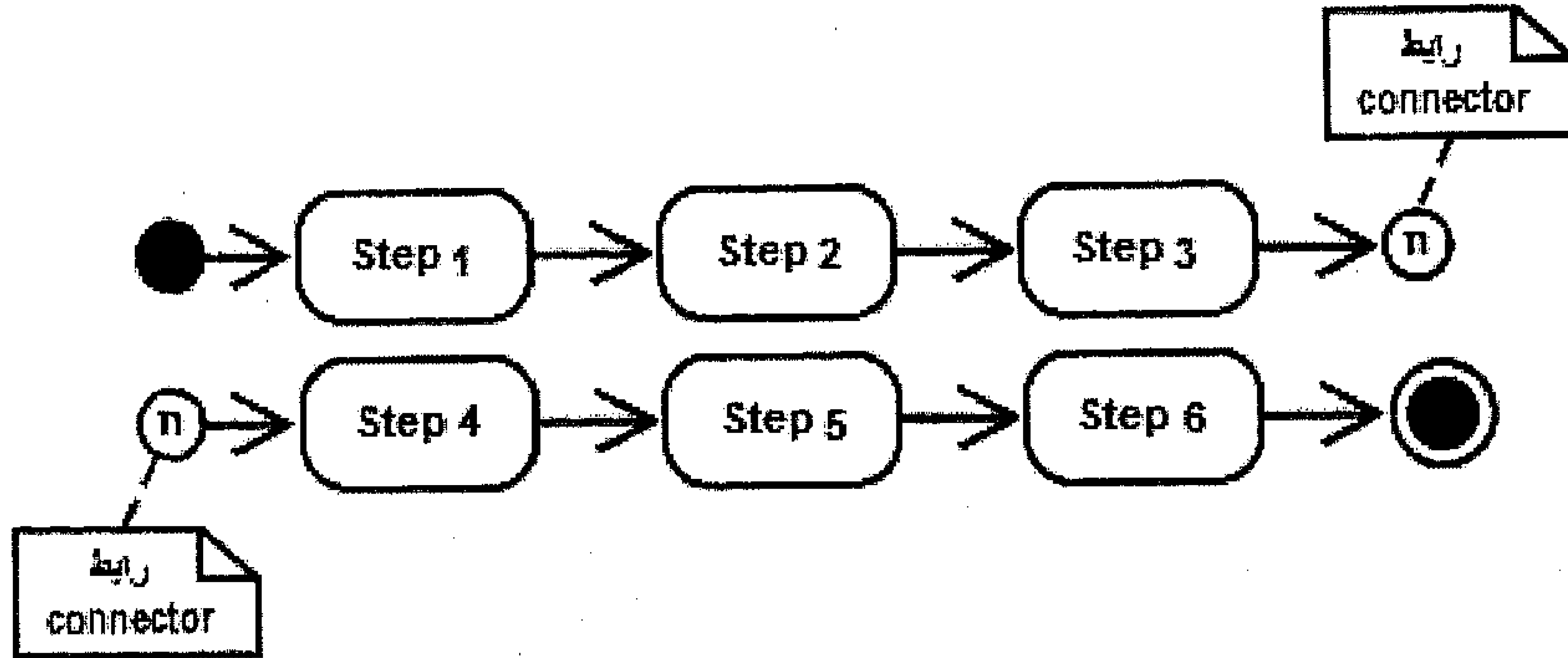
توفر مخططات النشاط العديد من الرموز الإضافية لنمذجة تشكيلة واسعة من العمليات. تبرز الأقسام التالية بعض الطرق المختصرة الملائمة لتبسيط مخططات النشاط. وللحصول على قائمة أشمل من هذه الترميزات انظر إلى الكتاب "UML 2.0 in a Nutshell (O'Reilly)".

٣-١٢-١ الروابط Connectors

إذا احتوى مخطط النشاط الكثير من الأفعال، قد تصادف خطوطاً طويلة ومتقاطعة تجعل المخطط صعب القراءة. وتساعد الروابط حينئذ في تبسيط المخطط.

تساعد الروابط على تفكيك المخططات بربط المسارات بواسطة الرموز بدلاً من الخطوط الصريحة. ويُرسم الرابط كدائرة حيث يُكتب اسمه بداخلها. عادة ما يتم إعطاء الروابط أسماء مؤلفة من حرف واحد. تم استعمال الاسم "n" للرابط الذي في الشكل رقم (٣-٢٦).

تأتي الروابط بشكل مزدوج: واحدة لها مسار دخول والثانية لها مسار خروج. ويتابع الرابط الثاني أينما توقف الرابط الأول. وبالتالي لا يتغير التدفق الذي في الشكل رقم (٣-٢٦) إذا كان عندنا مسار مباشر من الخطوة ٣ إلى الخطوة ٤.



شكل (٣-٢٦) يمكن أن تحسِّن الروابط من مقروئية مخطط نشاط ضخم.

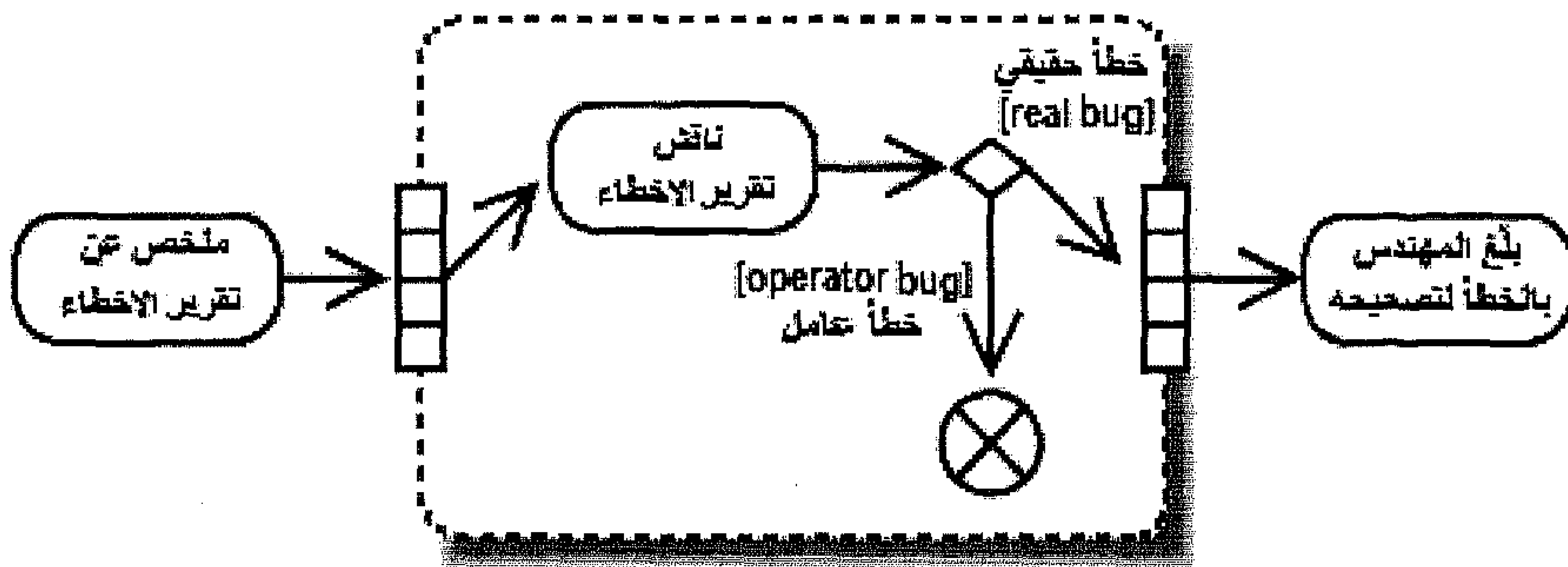


يجب الحذر مع الروابط: عند استعمال الكثير من الروابط المختلفة في المخطط الواحد، قد يواجه القارئ صعوبة في البحث عن طريف كل رابط.

٣-١٢-٢ مناطق التوسّع Expansion Regions

تشير مناطق التوسّع إلى تنفيذ الأفعال التي في داخلها على كل عنصر من مجموعة مدخلات. وعلى سبيل المثال، يمكن استعمال منطقة توسّع لنمذجة وظيفة برنامج تأخذ قائمة ملفات كمدخل لها وتبحث في كل ملف عن عنصر بحث محدد.

وترسم منطقة التوسّع كمستطيل كبير مدوّر الزوايا حيث تكون أضلعه منقطة ولديه أربعة صناديق مرصوصة على كلا جانبيه. وتمثل هذه الصناديق الأربعة مجموعات المدخلات والمخرجات (لكنها لا تشير ضمنياً إلى أن حجم المجموعة هو أربعة). ويظهر الشكل رقم (٣-٢٧) أنه تتم مناقشة كل من تقارير الخطأ التي في مجموعة المدخلات. وإذا كان الخطأ حقيقي، فيتم متابعة النشاط؛ وإلا فيتم استبعاد الخطأ وينتهي التدفق المعني به.



شكل رقم (٣-٢٧) تنفذ الأفعال التي في منطقة التوسّع على كل عنصر في المجموعة.

٣-١٣ ما هي الخطوة التالية؟

توجد مخططات أخرى في لغة النمذجة الموحدة تستطيع نمذجة السلوك الديناميكي للنظام مثل مخططات التتابع ومخططات الاتصال. وتركّز هذه المخططات على إظهار التفاعلات المفصّلة، مثل الكائنات

المشتركة في تفاعل ما، والطرق التي يتم استدعاؤها، وتتابع الأحداث. ويمكن إيجاد مخططات التتابع في الفصل السابع. وستغطى مخططات الاتصال في الفصل الثامن.

ومن المفيد قراءة الفصل الثاني عن حالات الاستخدام، إذا لم تكن قد قرأته بعد؛ لأن مخططات النشاط توفر وسيلة عظيمة لعرض تمثيل صوري لتدفق حالات الاستخدام.

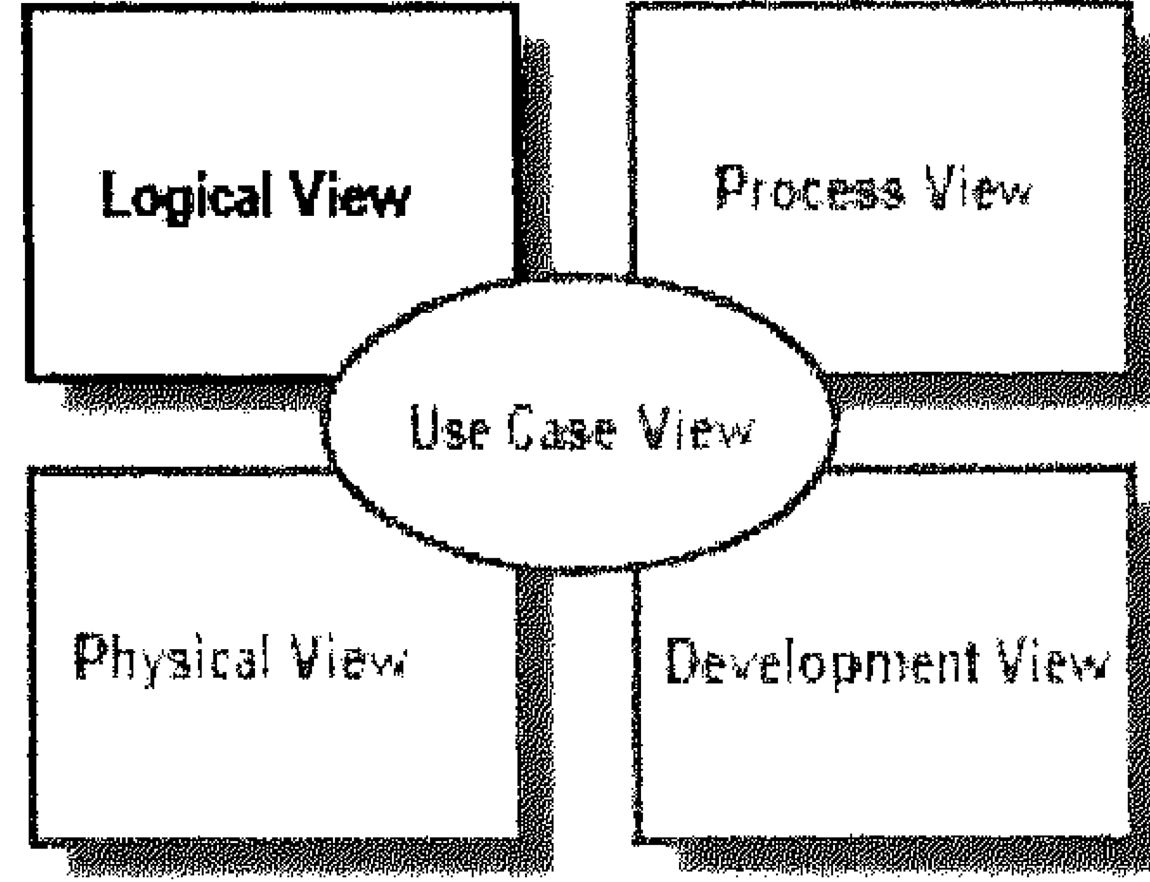
الفصل الرابع

نمذجة الهيكل المنطقي للنظام: تقديم الأصناف ومخططات الأصناف

MODELING A SYSTEM'S LOGICAL STRUCTURE: INTRODUCING CLASSES AND CLASS DIAGRAMS

تشكل الأصناف قلب أي نظام كائني التوجه؛ ويمكن استنتاج الشعبية الكبيرة لمخطط الأصناف من بين مخططات لغة النمذجة الموحدة. ويتألف هيكل النظام من مجموعة أجزاء يشار عادة إليها على أنها كائنات **objects**. وتصف الأصناف الأنواع المختلفة للكائنات التي يمكن تواجدها في النظام، وتظهر مخططات الأصناف تلك الأصناف والعلاقات التي بينها. (ويتم تغطية علاقات الأصناف في الفصل الخامس).

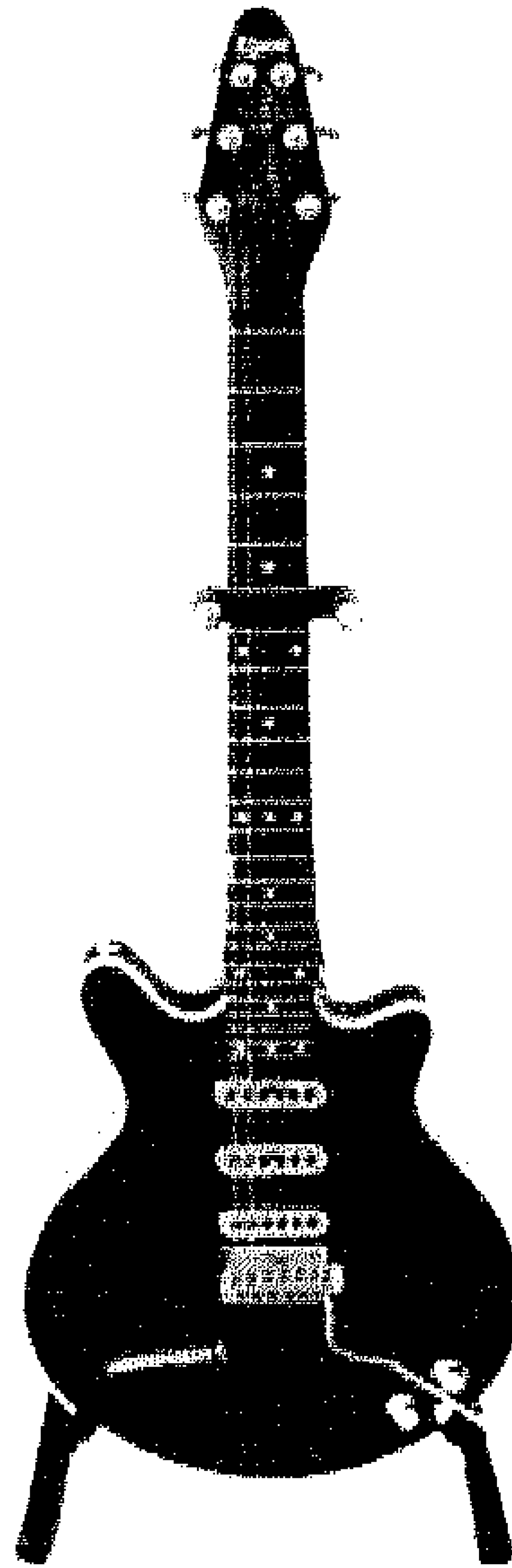
تصف حالات الاستخدام سلوك النظام كمجموعة أمور مهمة. وتصف الأصناف الأنواع المختلفة للكائنات الضرورية داخل النظام للتعامل مع تلك الأمور المهمة. وتشكل الأصناف جزءاً من المنظور المنطقي للنموذج، كما هو معروض في الشكل رقم (٤-١).



شكل رقم (٤-١) يحتوي المنظور المنطقي للنموذج على التوصيفات المجرّدة لأجزاء النظام، بما فيها الأصناف.

٤-١ ما هو الصنف؟ What is a Class?

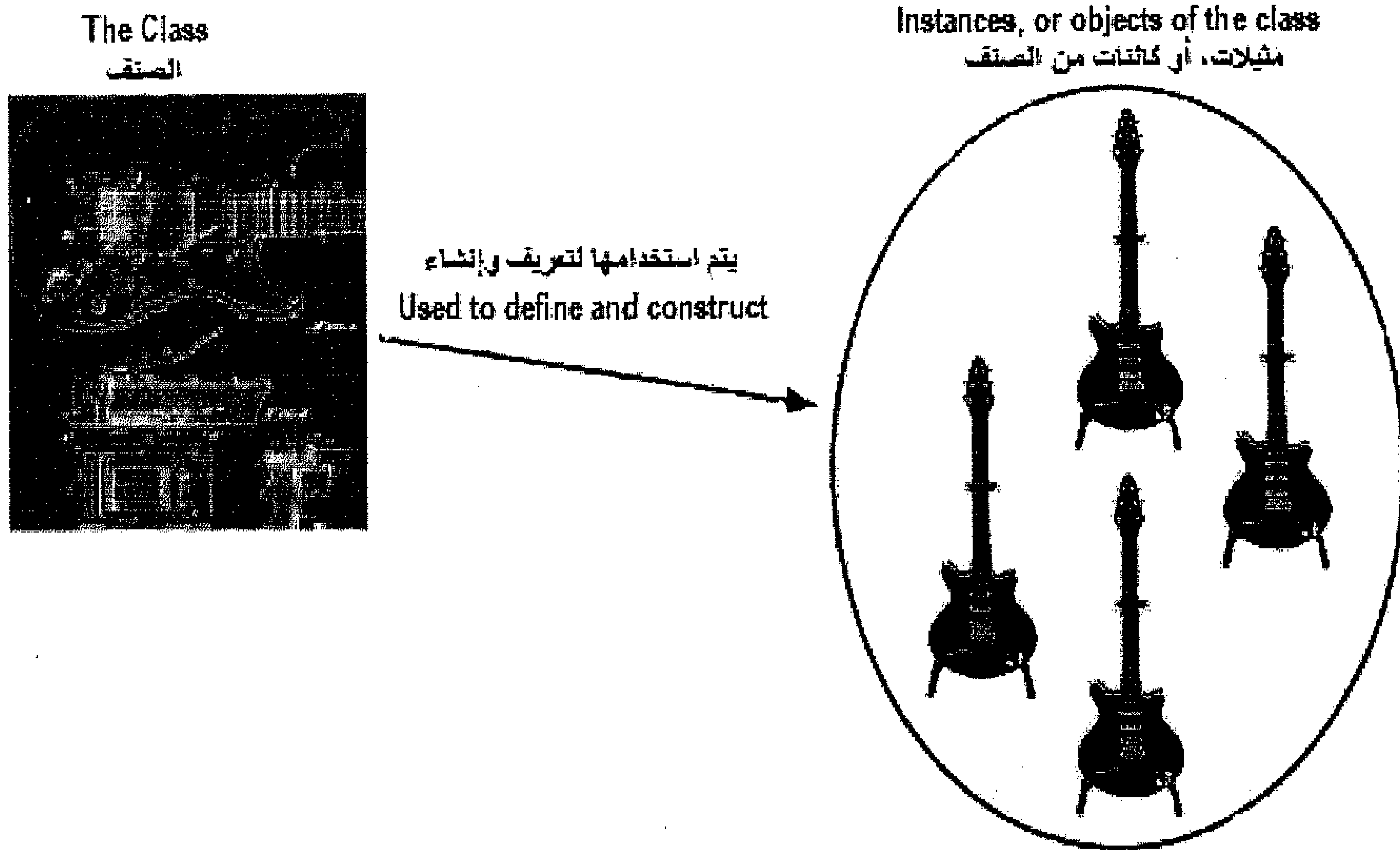
عند التفكير بجدية بمفهوم جديد كالأصناف، من المفيد البدء بمناظرة معينة. وسنستعمل المناظرة الخاصة بالقيثارات guitars (آلة موسيقية)، وقيثارتي المفضلة هي القيثارة Burns Brian May Signature (BMS)، المعروضة في الشكل رقم (٤-٢).



شكل رقم (٤-٢) إحدى قيثاراتي التي تشكل مثالاً جيداً عن كائن محدد.

وتشكل القيثارة التي في الشكل رقم (٤-٢) مثالاً واقعياً عن كائن محدد، يتمتع هذا الكائن بهوية identity: هي القيثارة التي أمتلكها. على أية حال، لن أزعم أنّ برنز Burns صنعت قيثارة واحدة فقط من هذا النوع وإنها كانت فقط لأجلي. فمن المؤكد إن شركة برنز قد صنعت المئات من هذا النوع من القيثارات أو بصيغة أخرى من هذا الصنف **class** من القيثارات.

ويشكل الصنف نوعاً لشيء ما. ويمكن التفكير بالصنف على أنه التصميم الذي يستعمل لصنع الكائنات، كما هو معروض في الشكل رقم (٤-٣).



شكل رقم (٤-٣) يعرّف الصنف الخصائص الرئيسية للقيثارة؛ ويمكن إنشاء أي عدد من كائنات القيثارة باستعمال هذا الصنف.

في هذه المناظرة، تشكل القيثارة BMS التي تصنعها شركة برنز مثالاً عن صنف قيثارة ما. وتعرف شركة برنز كيف تبني هذا النوع من

القيثارات من الصفر بالارتكاز على مخططها الأساسي. وكل قيثارة تم إنشاؤها باستعمال الصنف تتم الإشارة إليها على أنها مثيل **instance** أو كائن **object** من الصنف، لذلك تشكل القيثارة التي في الشكل رقم (٤-٢) مثيلاً للصنف **Burns BMS Guitar**.

يتضمن وصف الصنف قسميْن من المعلومات: معلومات الحالة state التي ستحتويها كائنات الصنف ومعلومات السلوك behavior الذي ستدعمه هذه الكائنات. وهذا ما يُميّز فكرة التوجه الكائني Object Oriented عن الأفكار الأخرى في تطوير النظام. ومع فكرة التوجه الكائني، يتم جمع الحالة والسلوك المحكمة العلاقة داخل تعريف الصنف، الذي يستعمل لاحقاً كنسخة كربونية blueprint لإنشاء الكائنات بواسطتها.

في حالة الصنف **Burns BMS Guitar**، يمكن أن تتضمّن حالة الصنف معلومات عن عدد خيوط هذه القيثارة وعن حالتها. وتسمى تلك الأجزاء من المعلومات بخصائص الصنف attributes. ونحتاج أيضا إلى معرفة ما يمكن أن تعمله القيثارة، ويتضمّن هذا سلوك القيثارة، مثل سلوك ضبط القيثارة وسلوك العزف عليها. ويتم وصف سلوك الصنف من خلال العمليات المختلفة التي يدعمها.

تشكل الخصائص و العمليات الأركان الأساسية في وصف الصنف (انظر إلى القسم ٤-٤ "حالة الصنف: الخصائص"). إنهما تمكنان الصنف من وصف مجموعة أجزاء ذات خصائص مشتركة داخل النظام، مثل الحالة المثلة بخصائص الصنف والسلوك المثل بعمليات الصنف (انظر إلى القسم ٤-٥ "سلوك الصنف: العمليات" الآتي لاحقاً في هذا الفصل).

٤-١-١ التجريد Abstraction

يحتوي تعريف الصنف على التفاصيل حول هذا الصنف والتي تكون مهمة للنظام الذي تقوم بنمذجته. على سبيل المثال، يمكن أن يوجد خدش أو أكثر على ظهر قيثارتي من النوع BMS، لكن إذا أنشأت صنفاً يمثل قيثارات BMS، فهل أحتاج إلى إضافة خصائص تمثل تفاصيل عن هذه الخدوش؟ أعتقد ذلك في حال استعمال هذا الصنف في ورشة تصليح؛ على أية حال، إذا كان هذا الصنف سيستعمل في نظام التصنيع فقط، يمكن إهمال تفاصيل الخدوش تفصيلاً بكل اطمئنان. ويتلخص مفهوم التجريد abstraction بالتخلص من التفاصيل غير المهمة في سياق محدد.

دعنا نلقي نظرة على مثال حول كيفية تغيير تجريد الصنف بالاعتماد على سياقه. إذا أنشأ برنز نموذجاً لنظامه الخاص بإنتاج القيثارات، فمن المرجح رغبته بإنشاء صنف **Burns BMS Guitar** ينمذج كيفية إنشاء قيثارة، والمواد المستعملة بالإنشاء وكيفية اختبارها. بالمقابل، إذا قام متجر قيثارات عالمي بإنشاء نموذج عن نظام بيع القيثارات، يمكن بالتالي للصنف **Burns BMS Guitar** احتواء المعلومات المهمة عن القيثارة فقط، مثل الرقم التسلسلي والسعر وربما أي تعليمات خاصّة بالمعالجة.

وغالباً ما يعتبر الحصول على مستوى التجريد الصحيح للنموذج أو حتى للصنف تحدياً حقيقياً. ركّز على المعلومات الضرورية للنظام بدلاً من التورط بالتفاصيل غير المهمة له. يصبح لدينا نقطة بداية جيدة عند تصميم أصناف النظام.



يعتبر التجريد شيئاً رئيسياً ليس بالنسبة لمخطط الأصناف فقط بل للنمذجة بشكل عام. فتعريف النموذج هو تجريد للنظام الذي يقوم بتمثيله. والنظام الواقعي هو الشيء الحقيقي؛ ويحتوي النموذج فقط على القدر الكافي من المعلومات ليشكل تمثيلاً دقيقاً للنظام الواقعي. في أكثر الحالات، يتجرد النظام من التفاصيل غير المهمة من أجل دقة التمثيل.

٤-١-٢ التغليف Encapsulation

قبل التعمق أكثر في تفاصيل الخصائص و العمليات و كيفية عمل الأصناف معاً، سيتم التركيز على الميزات الأكثر أهمية للأصناف والتوجه الكائني: التغليف **encapsulation**.

وفقاً للمنهجية كائنية التوجه في تطوير الأنظمة، يحتاج الكائن كي يكون كائناً إلى احتواء البيانات والعمليات operations ؛ البيانات عبارة عن الخصائص attributes والعمليات عبارة عن التعليمات التي تؤثر فيها. هذا هو الاختلاف الكبير بين المنهجية الكائنية التوجه والمنهجيات الأخرى في تطوير الأنظمة. ومع التوجه الكائني، يوجد مفهوم الكائن الذي يحتوي أو يُغلّف معاً البيانات و العمليات التي تعمل على تلك البيانات.

وبالإشارة إلى المناظرة السابقة عن القيثارة، يمكن أن يغلّف صنف قيثارة Burns BMS Guitar خيوطها وجسمها وعنقها، ومن المحتمل بعض الأمور الكهربائية الدقيقة التي لا يجب مسها. وتشكل أجزاء القيثارة خصائصها، ويمكن الوصول إلى بعض من هذه الخصائص من العالم الخارجي، مثل خيوطها، بينما تكون خصائص أخرى مخفية بعيداً، مثل الأمور الكهربائية. بالإضافة إلى تلك الخصائص، يحتوي الصنف Burns BMS Guitar على بعض العمليات التي تسمح للعالم الخارجي بالعمل على خصائص القيثارة. وكحد أدنى، يجب على صنف القيثارة أن يكون لديه

علـى الأقـل عمليـة اسمهـا أعـزف **play**، كـي يصـبح بإمكـان كائناتـه العزف، لكن قد تكون العمليات الأخرى، مثل نظّف **clean** وربما حتى الخدمات الكهربائية **serviceElectrics**، أيضاً مُغلّفة ويوفرها الصنف.

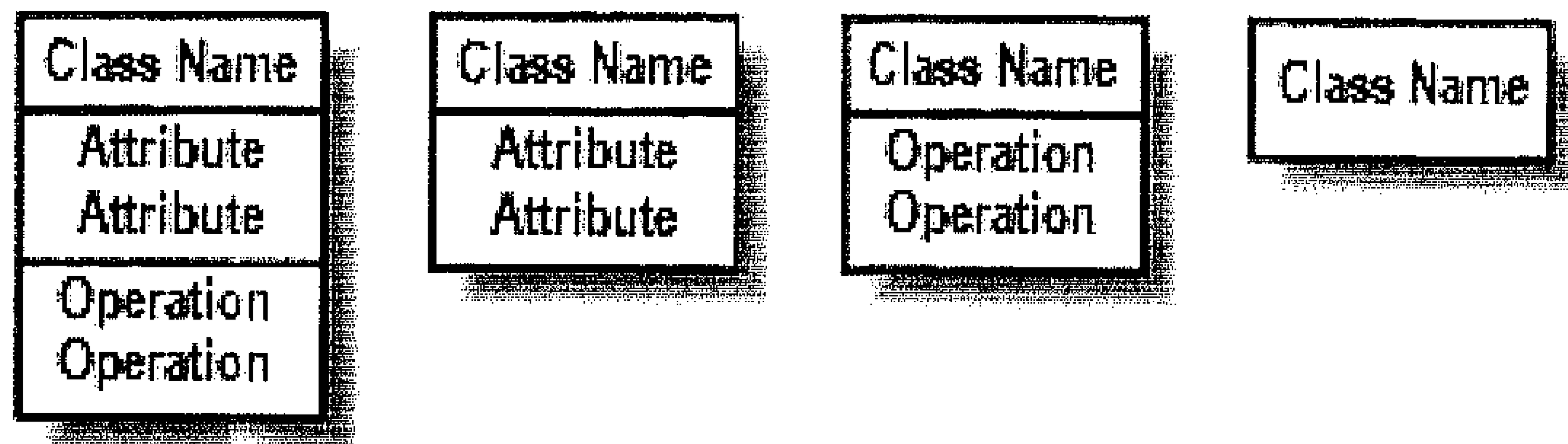
وربمـا يشـكل تغليف العمليات و البيانات داخل الكائن الجزء الأقوى و الأفيَد للمنهجية الكائنية التوجه في تصميم النظام. ويسمح التغليف للصنف بإخفاء التفاصيل الداخلية عن العالم الخارجي، وهذا فيما يتعلق بكيفية عملها، مثل الأمور الكهربائية المتعلقة بصنف القيثارة، ويسمح التغليف أيضاً بكشف العمليات و البيانات التي يسمح صنفها بالوصول المباشر إليها.

ويعتبر التغليف أمراً مهماً؛ لأنه يسمح للصنف بتغيير الطريقة التي يعمل بها داخلياً، ولن يكون لتلك التغييرات تأثير على كيفية التفاعل مع الصنف، ما دامت تلك الأمور الداخلية غير مرئية لبقيّة النظام. وهذه ميزّة مفيدة للمنهجية الكائنية التوجه، لأنه مع الأصناف المعرفة بشكل صحيح، يجب ألاّ تسبب التغيرات البسيطة الخاصة بكيفية عمل الأصناف داخلياً من إيقاف النظام.

٤-٢ البدء مع الأصناف في لغة النمذجة الموحدة
Getting Started with Classes in UML

لقد رأينا تعريف الأصناف وكيفية تعزيزها للفوائد الأساسية لمنهجية التوجه الكائني في تطوير الأنظمة من خلال التجريد والتغليف. وقد حان الوقت لإلقاء نظرة على كيفية تمثيل الأصناف في لغة النمذجة الموحدة.

ويرسم الصنف في UML على شكل مستطيل مقسم إلى ثلاثة أقسام. يحتوي القسم الأعلى على اسم الصنف، ويحتوي القسم الأوسط على الخصائص أو المعلومات التي يحتويها الصنف، ويحتوي القسم الأخير على العمليات الممثلة للسلوك الذي يقدمه الصنف. إن قسمي الخصائص والعمليات هي أقسام اختيارية، كما هو معروض في الشكل رقم (٤-٤). إذا كان قسمي الخصائص والعمليات غير ظاهرين، فهذا لا يعني بالضرورة أنهما فارغان، بل يعني أنه ربما يكون المخطط أسهل للفهم بمجرد إخفاء تلك المعلومات.



شكل رقم (٤-٤) تقديم أربع طرق مختلفة لعرض صنف باستعمال ترميز UML.

يؤسس اسم الصنف نوعاً للكائنات التي سيتم إنشاؤها بالارتكاز عليه. يعرض الشكل رقم (٤-٥) صنفين من نظام إدارة المحتوى في الفصل الثاني، ويعرّف الصنف حساب مدونة **BlogAccount** معلومات حسابات المستخدم التي سيحتفظ بها النظام، ويعرّف الصنف تدوينة أو مدخل في مدونة **BlogEntry** معلومات تدوينة مستخدم في مدونته.



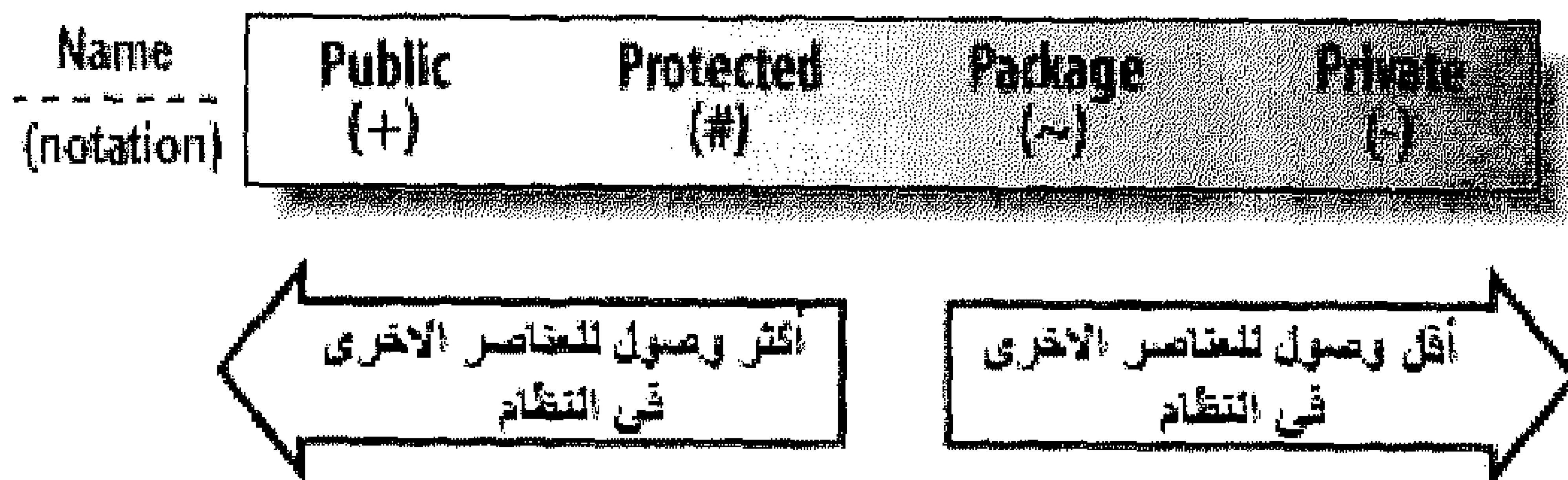
شكل رقم (٤-٥) صنفان من الكائنات تم التعرف عليهما في نظام إدارة المحتوى.

تستعمل مخططات التفاعل، المغطات من الفصل السابع حتى الفصل العاشر، لعرض كيفية عمل الكائنات (مثيلات الأصناف) معاً عند تشغيل النظام.

٤-٣ الرؤية Visibility

كيف يكشف الصنف بشكل اختياري عن عملياته وبياناته إلى الأصناف الأخرى؟ يتم ذلك باستعمال الرؤية **visibility** (أو ما يسمى بمحددات الوصول إلى أعضاء الأصناف). وعند تطبيق خصائص الرؤية يتم التحكم بعملية الوصول إلى الخصائص والعمليات وحتى كامل الأصناف للالتزام عملياً بمفهوم التغليف. (انظر سابقاً إلى "التغليف" في هذا الفصل للمزيد من المعلومات عن سبب كونه سمة مفيدة في التصميم الكائني التوجه للنظام).

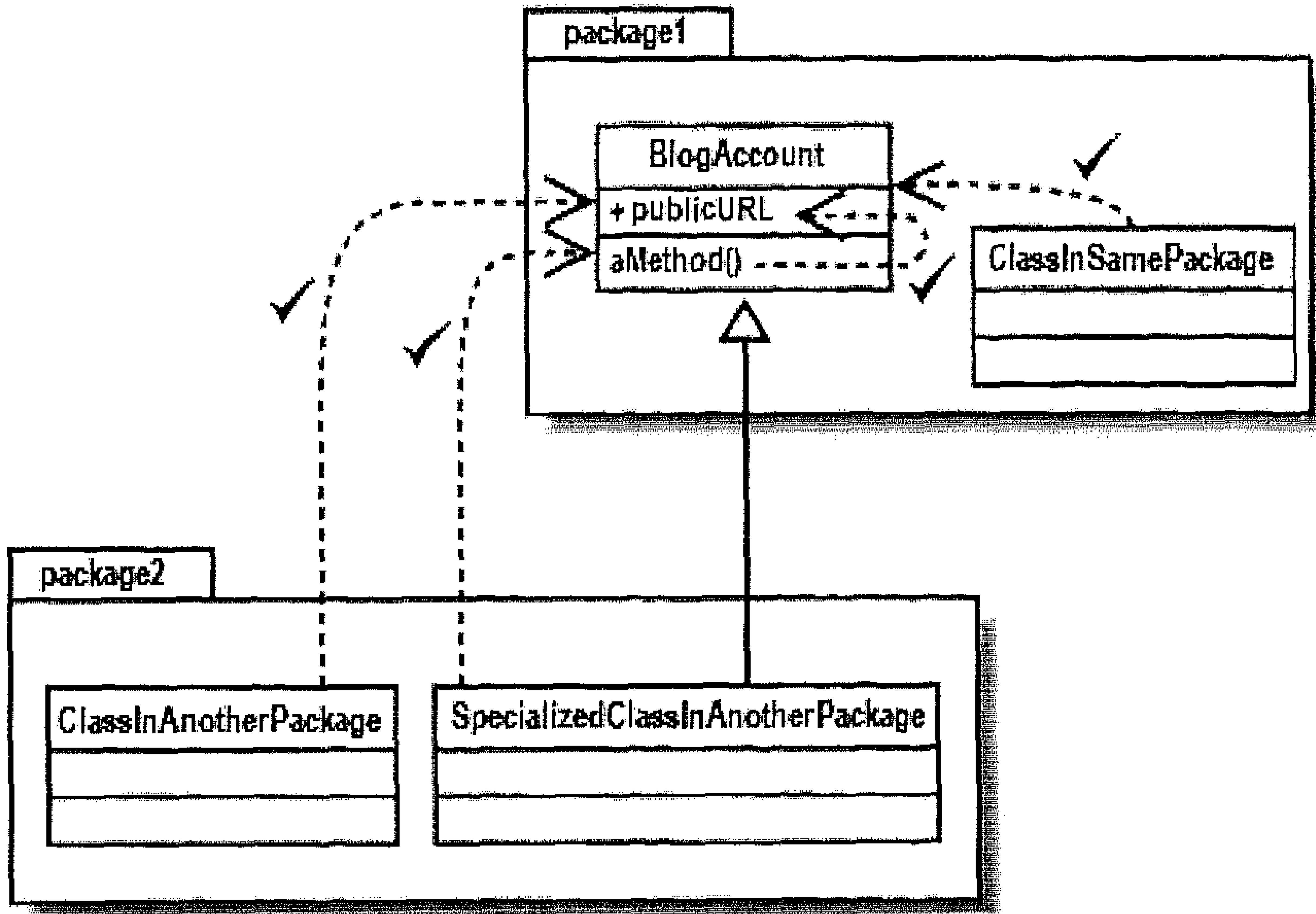
هناك أربعة أنواع مختلفة من الرؤية يمكن تطبيقها على عناصر نموذج UML، كما هو معروض في الشكل رقم (٤-٦). وعادة ما تستعمل ميزات الرؤية للتحكم بالوصول إلى الخصائص والعمليات وأحياناً إلى الأصناف (انظر إلى قسم "الحزم packages" في الفصل الثالث عشر للمزيد من المعلومات عن الرؤية الخاصة بالصنف).



شكل رقم (٤-٦) تصنيفات الرؤية الأربعة المختلفة في UML.

٤-٣-١ الرؤية العامة Public Visibility

نبدأ مع خاصية الرؤية الأكثر سماحاً بالوصول، يتم تحديد الرؤية العامّة باستعمال رمز الزائد (+) قبل الخاصية أو العملية المرتبطة به (انظر الشكل رقم ٤-٧). ويتم التصريح عن خاصية أو عملية على أنها عامة للسماح بالوصول المباشر إليها من أي صنف آخر.



شكل رقم (٤-٧) باستعمال الرؤية العامّة، يمكن لأي صنف داخل النموذج من الوصول إلى الخاصية publicURL.

تقوم مجموعة الخصائص والعمليات التي يتم التصريح عنها عامة في الصنف بإنشاء الواجهة interface العامة للصنف. وتتألف الواجهة العامّة للصنف من الخصائص والعمليات التي يمكن الوصول إليها واستعمالها من قِبَل الأصناف الأخرى. وهذا يعني أن الواجهة العامّة هي الجزء من الصنف الذي تعتمد عليه الأصناف الأخرى إلى أبعد حد. ومن المهم تقليل التغيير في

الواجهة العامة للصنف بالقدر المستطاع لمنع التغييرات غير الضرورية حيثما يتم استعمال الصنف.

الخصائص العامّة Public Attributes

هل من المفيد السماح بالخصائص العامة؟ ينصرف عديد من مصممي التوجه الكائني عن استعمال الخصائص العامة، ويعتبر فتح خصائص الصنف على باقي النظام مثل تعريض بيتك لأي شخص من الشارع من دون الفرض عليه التشاور معك قبل دخوله، وتوجد فرص كثيرة لسوء استخدام الخصائص بهذه الحالة.

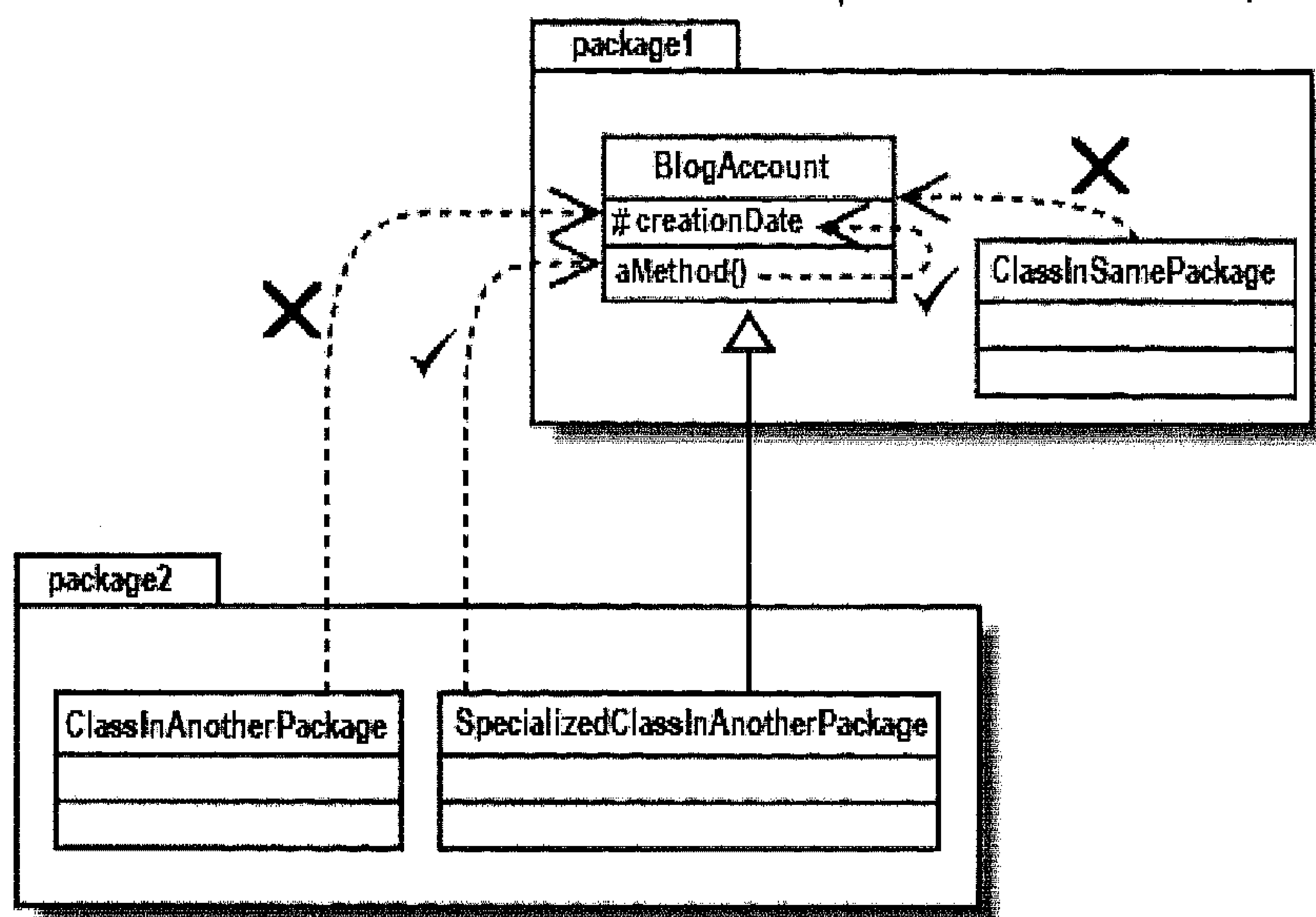
ومن الأفضل – عادة – تفادي الخصائص العامة، لكن يوجد دائماً استثناءات للقاعدة. أحد الأمثلة على قبول استعمال الخاصيّة عامة هو عندما تكون الخاصيّة ثابتة القيمة constant لإتاحتها للاستعمال من قِبَل عدد من الأصناف المختلفة. والثوابت هي التي تعطى قيمة أولية لا يمكن تغيرها لاحقاً. ويتم منح الخصائص التي تعمل كالثوابت ميزة للقراءة فقط readOnly (انظر إلى "ميزات الخصائص Attribute Properties"). في هذه الحالة، لا توجد خطورة جراء فتح الخاصية على باقي النظام بسبب عدم التمكن من تغيير قيمتها.

٤-٣-٢ الرؤية المحمية Protected Visibility

يتم تحديد الخصائص و العمليات المحمية باستعمال الرمز (#)، و هي تكون أكثر رؤية لبقيّة النظام من الخصائص و العمليات الخاصّة private، لكنها تكون أقل رؤية من الخصائص و العمليات العامة. ويمكن الوصول إلى العناصر المصرح عنها محميّة داخل الصنف من داخل طرق هذا الصنف و أيضاً من داخل طرق الأصناف التي ترث منه. ولا يمكن الوصول إلى العناصر المحمية من قبل الأصناف التي لا ترث من صنفها سواء كانت هذه الأصناف في نفس الحزمة أم في حزمة أخرى،

كما هو معروض في الشكل رقم (٤-٨). (انظر إلى الفصل الخامس للمزيد من المعلومات عن علاقات الوراثة inheritance بين الأصناف).

وتكمن أهمية الرؤية المحمية في السماح للأصناف الوارثة (المخصصة specialized) بالوصول إلى خاصيّة أو عملية في الصنف الموروث (الأساسي base) من دون فتح تلك الخاصيّة أو العملية على كامل النظام. ويكون استعمال الرؤية المحمية مثل القول، "هذه الخاصيّة أو العملية هي مفيدة داخل صنفي و الأصناف التي ترث أو توسع صنفي، لكن لا يجب استعمالها من قِبَل أحد غيرهم ".



شكل رقم (٤-٨) يمكن لأي طُرُق (عمليات) methods في الصنف BlogAccount أو الأصناف التي ترث منه أن تصل إلى الخاصية المحمية createDate.



تقوم لغة البرمجة جافا بتشويش المسألة أكثر قليلاً من خلال السماح بالوصول إلى الأجزاء المحمية في الصنف من أيّ صنف آخر في نفس الحزمة. ويكون هذا بمثابة جمع إمكانية الوصول للرؤيتين محمية وحزمية معاً، والتي يتم تغطيتها في القسم القادم.

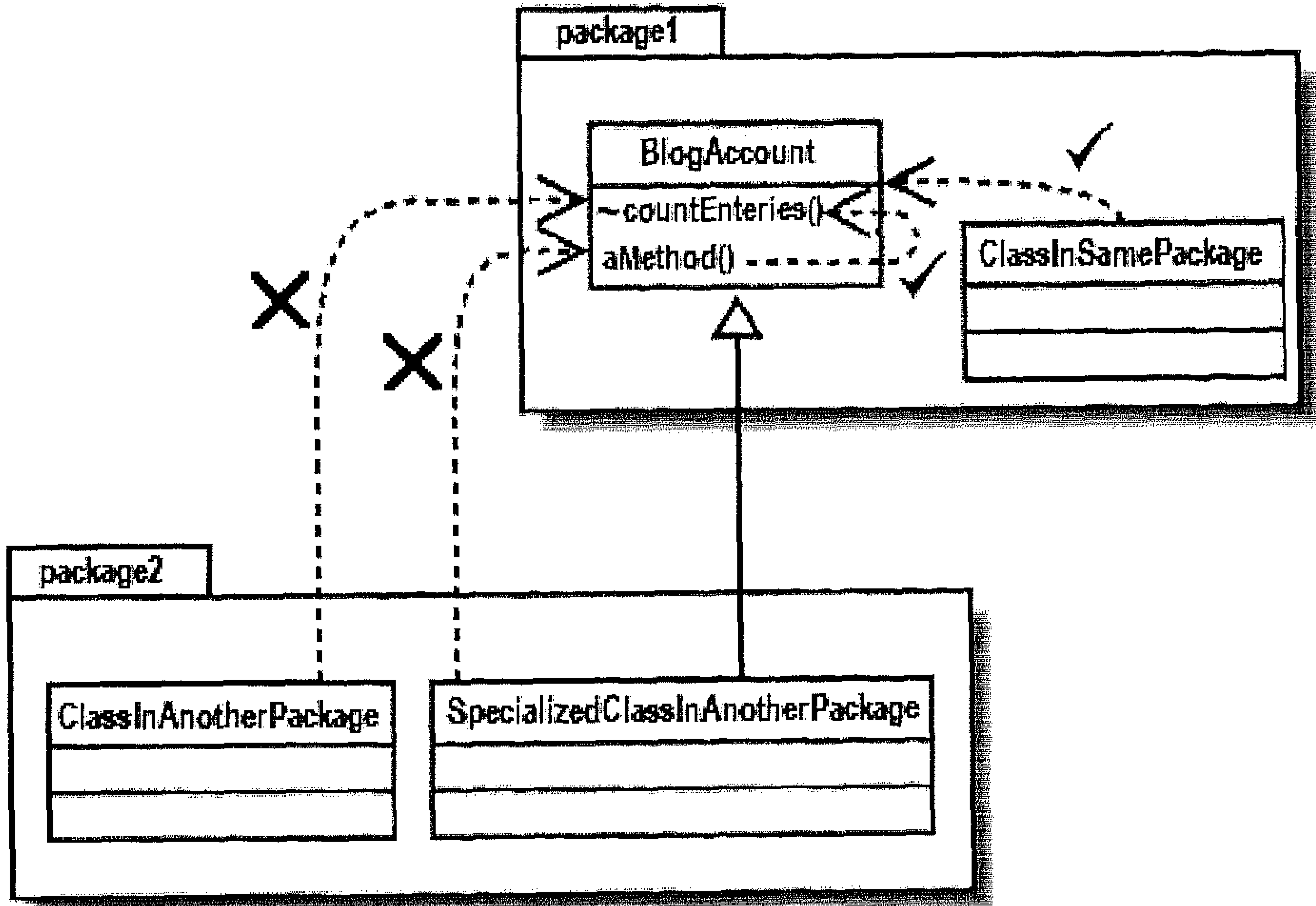
٤-٣-٣ الرؤية الحزمية Package Visibility

يتم تحديد الرؤية الحزمية باستعمال الرمز تِلدة " (~) tilde"، وتقع الرؤية الحزمية عند تطبيقها على الخصائص والعمليات بين الرؤية المحمية والرؤية الخاصة. كما كنت تتوقع، تشكل الحُزَم العامل الأساسي في تحديد الأصناف التي يمكنها رؤية الخاصيات أو العمليات المصرح عنهم باستعمال الرؤية الحزمية.

وتوجد قاعدة بسيطة جداً: عند استعمال الرؤية حزمية لأي خاصيّة أو عملية في الصنف، يمكن بالتالي لأي صنف في نفس الحزمة من الوصول مباشرة إلى تلك الخاصية أو العملية، كما هو معروض في الشكل رقم (٤-٩). ولا تستطيع الأصناف التي خارج الحزمة الوصول إلى الخصائص أو العمليات ذات الرؤية حزمية حتى إذا كانت هذه الأصناف ترث من الصنف المصرح عنها فيه. بشكل عملي، وتفيد الرؤية الحزمية في التصريح عن مجموعة خصائص و طُرُق تريد استعمالها فقط داخل أصناف الحزمة.

على سبيل المثال، إذا قمت بتصميم حزمة أصناف خدماتية، وأردت إعادة استعمال العمليات بين تلك الأصناف ولكن من دون كشفها على باقي النظام، عليك بالتالي التصريح عن تلك العمليات باستعمال الرؤية الحزمية لتكون داخلية في الحزمة. ويمكن كشف أي وظيفة في الأصناف الخدماتية لباقي التطبيق بالتصريح عنها باستعمال الرؤية عامة.

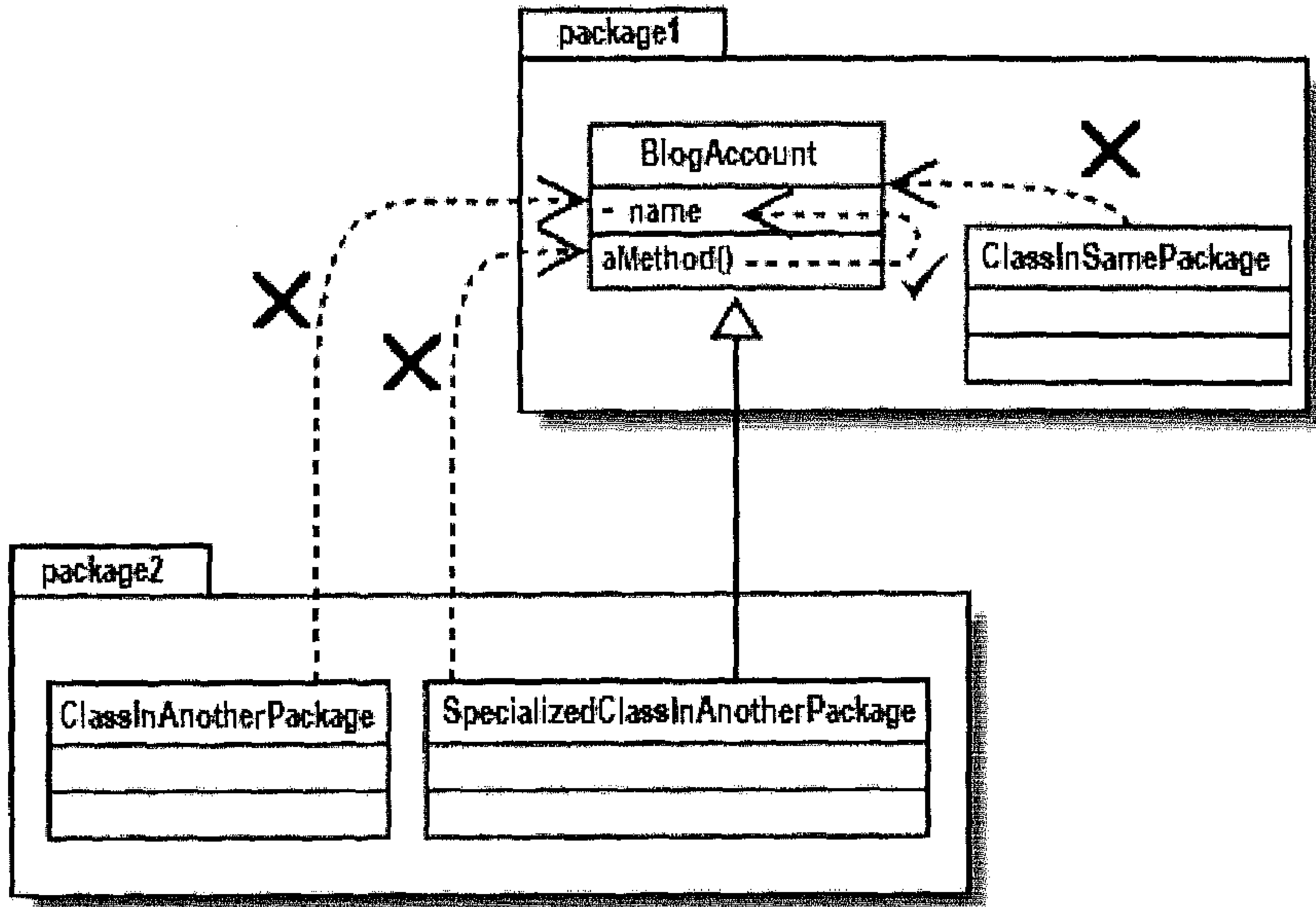
انظر إلى: "مخططات الحُزَم" في الفصل الثالث عشر للمزيد عن كيفية تحكم الحُزَم برؤية العناصر مثل الأصناف.



شكل رقم (٤-٩) يمكن استدعاء العملية countEntries من أي صنف في نفس الحزمة مثل الصنف ClassInSamePackage أو من أي طريقة داخل الصنف BlogAccount نفسه.

٤-٣-٤ الرؤية الخاصة Private Visibility

تأتي الرؤية الخاصة عند آخر خط مقياس الرؤية بلغة النمذجة الموحدة. وتشكل الرؤية الخاصة النوع الأكثر محدودية من بين تصنيفات الرؤية، ويتم تحديدها باستعمال الرمز سالب (-) قبل الخاصية أو العملية. يمكن فقط للصنف الذي يحتوي على العناصر الخاصة من رؤية واستعمال البيانات المخزنة في الخصائص الخاصة أو استدعاء العمليات الخاصة، كما هو معروض في الشكل رقم (٤-١٠).



شكل رقم (٤-١٠) تستطيع الطريقة **aMethod** الوصول إلى الخاصية الخاصّة **name** لأنهما في نفس الصنف **BlogAccount**، لكن لا تستطيع طرق الأصناف الأخرى رؤية **name**.

تفيد الرؤية الخاصّة عند الحاجة إلى تعريف خاصية أو عملية لا تريد لأي جزء آخر من النظام أن يعتمد عليها. وقد تكون هذه الحالة إذا عزمت تغيير خاصيّة أو عملية في وقت لاحق لكنك لا تريد تغيير الأصناف الأخرى التي تصل إليها.

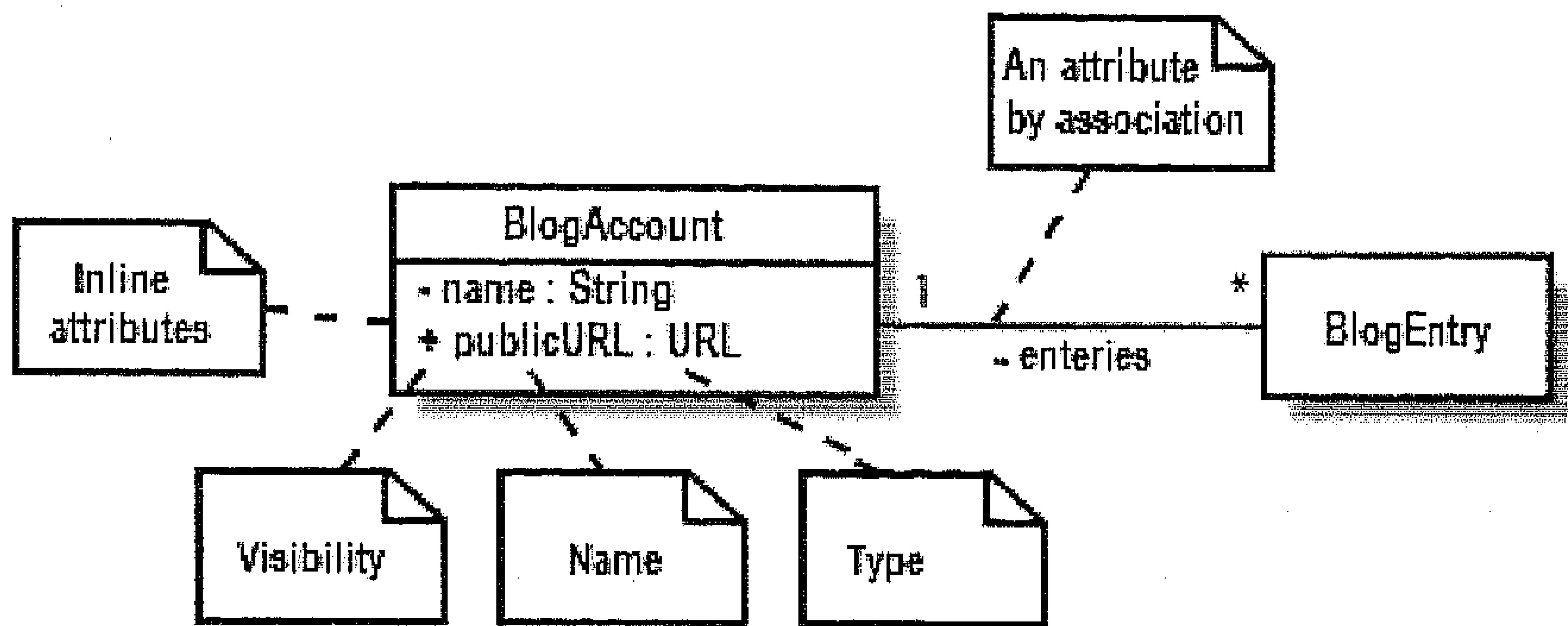


هناك قاعدة مجرّبة مقبولة وشائعة الاستعمال، يجب دائماً استعمال الرؤية الخاصة للخصائص، ويتم فتحها للوصول المباشر إليها فقط في الحالات المفرطة من خلال استعمال نوع رؤية أكثر انفتاحاً. والحالة الاستثنائية لهذه القاعدة هي عند الحاجة إلى مشاركة خاصيّة في الصنف مع الأصناف التي ترث منه. في هذه الحالة، ومن الشائع استعمال الرؤية المحمية. وفي الأنظمة الكائنية التوجه المصمّمة بشكل جيد، عادة ما تكون الخصائص خاصّة أو محمية، لكن نادراً جداً ما تكون عامة.

٤-٤ حالة الصنف: الخصائص
Class State: Attributes

تشكل خصائص الصنف عينة من المعلومات التي تمثل حالة كائن. ويمكن تمثيل هذه الخصائص في مخطط الأصناف إما بوضعها داخل القسم الخاص بها في مستطيل الصنف، حيث تُعرف بالخصائص الداخلية الضمنية inline، أو بواسطة الشراكة association مع صنف آخر، كما هو معروض في الشكل رقم (٤-١١). ولقد تم تغطية الشراكات بشكل أوسع في الفصل الخامس.

لا تهم كيفية الإعلان عن الخاصيّة داخلية inline أو شراكة. وعادة ما يكون للخاصية على الأقل توقيعاً signature يحتوي على نوع رؤيتها واسمها ونوعها البياني. بالرغم من أن اسم الخاصية هو الجزء الوحيد من توقيعها الذي يجب تواجده قطعياً في الصنف كي يكون صحيحاً.



شكل رقم (٤-١١) يحتوي الصنف **BlogAccount** على الخصائص الداخلية **name** و **publicURL**، بالإضافة إلى الخاصيّة **enteries** المُدرجة بواسطة الشراكة بين الصنفين **BlogAccount** و **BlogEntry**.

٤-٤-١ الأسم والنوع Name and Type

يمكن أن يتكون أسم الخاصية من أي مجموعة حروف، لكن لا يمكن أن تأخذ خاصيتان نفس الاسم في نفس الصنف. ويمكن أن يختلف النوع البياني للخاصية بالاعتماد على كيفية برمجة الصنف في النظام، لكن عادة ما يكون النوع البياني إما صنفاً، مثل صنف سلاسل الرموز String، أو أحد أنواع البيانات البدائية primitive type، مثل نوع الأعداد الصحيحة int في لغة جافا.

في الشكل رقم (٤-١١)، تم التصريح عن الخاصية name باستعمال الرؤية الخاصة (من خلال استعمال الرمز سالب (-) في بداية التوقيع)، وتم بعد النقطتين العموديتين (:) تحديد نوعها البياني على أنه الصنف String. ولخاصية الشراكة entries أيضاً الرؤية الخاصة، وهي تمثل عدداً من مثيلات الصنف BlogEntry بسبب تلك الشراكة.

اختيار أسماء الخصائص Choosing Attribute Names

تذكر أن أحد الأهداف الأساسية لنمذجة النظام هو تبليغ تصميمك إلى الآخرين. وعند اختيار أسماء للخصائص والعمليات والأصناف والحزم، تأكد بأن الاسم يصف بدقة المسمّى. وعند تسمية الخصائص، من المفيد محاولة اختراع اسم يصف المعلومات التي تمثلها الخاصية.

إضافة إلى ذلك، عند برمجة الصنف في لغة برمجة محددة، تأكد من موافقة الاسم المستعمل لاصطلاحات وقواعد تلك اللغة. في لغة جافا، ومن الشائع استعمال أحرف كبيرة **uppercase** لبداية كل كلمة في أسماء الأصناف، مثل **BlogAccount**، بينما تسمّى حزم جافا عادة باستعمال أحرف صغيرة حصرياً **lowercase** (انظر إلى الفصل الثالث عشر).

إذا أردنا برمجة الصنف BlogAccount في الشكل رقم (٤-١١) كصنف جافا في البرنامج، ستشبه شفرة مصدره شيئاً مثل الشفرة المعروضة في المثال رقم (٤-١).

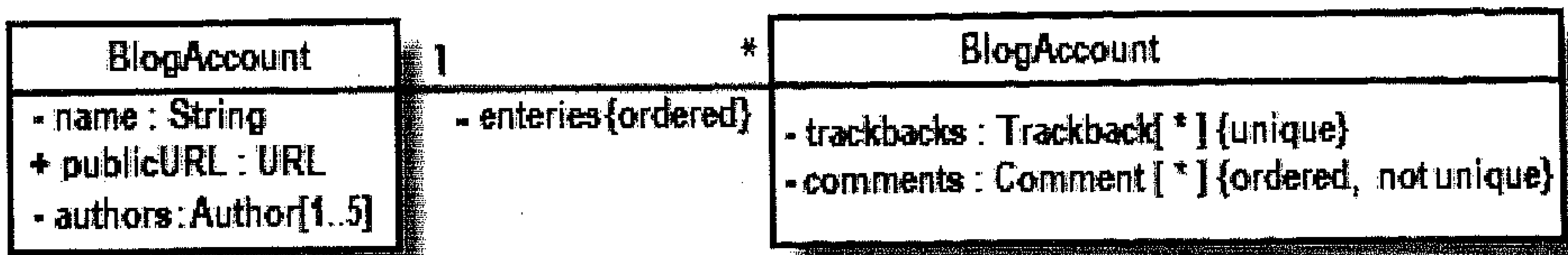
مثال رقم (٤- ١) الخصائص **inline** والتي بواسطة الشراكة في جافا.

```
public class BlogAccount {
// inline من المثال ٤- ١١ الخاصتين //
private String name;
private URL publicURL;
// entries الخاصيّة الفردية بواسطة الشراكة، المعطاة الاسم //
private BlogEntries [ ] entries;
// ...
}
```

من الواضح جداً كيفية برمجة الخاصيتين inline في صنف جافا BlogAccount؛ الخاصية name هي مجرد سلسلة رموز جافا من النوع الصنف String والخاصية publicURL هي كائن جافا من نوع الصنف URL. والخاصية entries أكثر أهمية بقليل لأنها مُدخلة بواسطة الشراكة. (لقد تم تغطية الشراكات والعلاقات بين الأصناف في الفصل الخامس).

٤-٤-٢ التعدّدية Multiplicity

تمثل أحياناً الخاصية أكثر من كائن واحد. وفي الحقيقة، يمكن أن تمثل الخاصية أي عدد من الكائنات التي من نوعها؛ ويكون هذا في البرمجة بالتصريح عن الخاصية بأنها مصفوفة array. وتسمح التعدّدية بتحديد تمثيل الخاصية فعلياً لمجموعة كائنات، ويمكن تطبيقها على الخصائص inline وخصائص الشراكة معاً، كما هو معروض في الشكل رقم (٤-١٢).



شكل رقم (٤-١٢) تطبيق أشكال مختلفة لتعدّدية الخاصية على خصائص الصنفين **BlogAccount** و **BlogEntry**.

في الشكل رقم (٤-١٢)، كل خصائص تقفيات الأثر للوراء trackbacks و التعليقات comments والكتبة authors تمثل مجموعات كائنات. يحدد رمز النجمة (*) في نهاية الخاصيتين trackbacks وcomments أنه يمكن أن تحتويا على التوالي على أي عدد من كائنات الصنفين Trackback و Comment. أما الخاصية authors فهي أكثر تحديداً بعض الشيء؛ لأنها توضح أنها تحتوي من كاتب واحد إلى خمسة كُتَّاب.

تملك الخاصية entries المدخلة باستعمال شراكة بين الصنف BlogAccount و الصنف BlogEntry قيمتين للتعددية تم تحديدهما عند طرفي الشراكة. ويشير رمز التعددية (*) الموجود عند طرف الشراكة من ناحية الصنف BlogEntry إلى تخزين أي عدد من كائنات BlogEntry في الخاصية entries داخل الصنف BlogAccount. وتشير التعددية ١ الموجودة عند الطرف الآخر للشراكة إلى ارتباط كل كائن BlogEntry في الخاصية entries بكائن BlogAccount واحد وواحد فقط.

نلاحظ أيضاً أن للخصائص trackbacks و comments و entries صفات إضافية تقوم بوصف أدق لمعنى التعددية المتعلق بالخصائص. وتمثل الخاصية trackbacks أي عدد من كائنات الصنف Trackback، لكن تم أيضاً تطبيق صفة التعددية فريدة unique عليها. تجبر الصفة فريدة على عدم تكرار نفس الكائن Trackback في المصفوفة. ويشكل هذا قيداً معقولاً لأننا لا نريد لتدوينة في مدونة أخرى من التشابك بالمراجع أكثر من مرة واحدة مع تدوينة من تدويناتنا؛ وإلا فتصبح القائمة (المصفوفة) Trackbacks غير منظمة.

وتكـون كـل الخصـائص ذات التعددية فريـدة بشـكل افتراضـي. وهـذا يعـني أنـه لا يتكـرر نفـس الكـائن في مجموعـة الكتبـة المحـددة بالخاصـية authors في الصـنف BlogAccount، كمـا هـي الحـال بالنسـبة للخاصـية trackbacks في الصـنف BlogEntry المصـرح عنهـا بأنهـا فريـدة. ويكـون لهـذا معنـى بسـبب تحديـده إمكانيـة أخـذ الكـائن BlogAccount حتـى خمسـة كُتَّـاب مختلفين؛ علـى أيـة حـال، لـيس هنـاك معنـى لتكـرار تحديـد نفـس الكاتـب أنـه يعمـل علـى المدونـة! إذا أردنـا تحديـد السـماح بـالتكرار، فنحتـاج بالتـالي إلى اسـتعمال الصـفة غير فريـدة not unique، كما تم ذلك مع الخاصية comments في الصنف BlogEntry.

الصفة الأخيرة المتعلقة بتعددية الخاصية هـي الصفة مرتبـة ordered. بالإضافة إلى عدم كونها فريـدة، تحتـاج الكـائنـات المسـتخدمة بالخاصـية comments في الصـنف BlogEntry إلى أن تكـون مرتبـة. وتسـتعمل الصـفة ordered في هذه الحالة لتحديد أن كل كائنات الصنف Comment مخزّنة بترتيب محـدد، ومرتبـة علـى الأغلـب حسـب إضـافتها إلى BlogEntry. وإذا كنت لا تهتم بأمر ترتيب تخزين الكـائنات داخل خاصية ذات تعددية، فلا تستخدم الصفة مرتبة ببساطة.

٤-٤-٣ ميزات الخاصية Attribute Properties

بالإضافة إلى الرؤية و الاسم الفريد و النوع البياني، هناك مجموعة ميزات يمكن تطبيقها على الخصائص لوصف ميزات خاصية بالكامل.

بالرغم من أن الوصف الكامل للأنواع المختلفة لميزات الخصـائص قد يكـون خـارج إطـار هـذا الكتـاب - يمكننـا القـول إن بعـض هـذه

الميزات نادرة الاستخدام عملياً - ومن الأفضل النظر إلى ميزات الخاصية الأكثر شعبية كميزة للقراءة فقط readOnly.

إذا طبقت ميزة للقراءة فقط readOnly على الخاصية، كما هو معروض في الشكل رقم (٤-١٣)، فلا يمكن تغيير قيمة الخاصية بعد تحديدها لأول مرة.

وإذا كان الصنف ContentManagementSystem سيبرمج بلغة جافا، سيتم إعطاء الصفة نهائية final للخاصية createdBy، كما هو معروض في المثال رقم (٤-٢).

ContentManagementSystem
- createdBy : String = "Adam Cook Software Corp." {readOnly}

شكل رقم (٤- ١٣) تم إعطاء الخاصية **createdBy** قيمة أولية افتراضية و ميزة للقراءة فقط **readOnly** كي لا يمكن تغيير الخاصية طيلة فترة حياة النظام.

مثال رقم (٤-٢) يشار إلى الصفة نهائية في جافا كالثوابت لأنها تحتفظ بنفس القيمة الثابتة المهدة بها طيلة عمرها.

```
public class ContentManagementSystem {
 private final String createdBy = "Adam Cook Software Corp.";
}
```

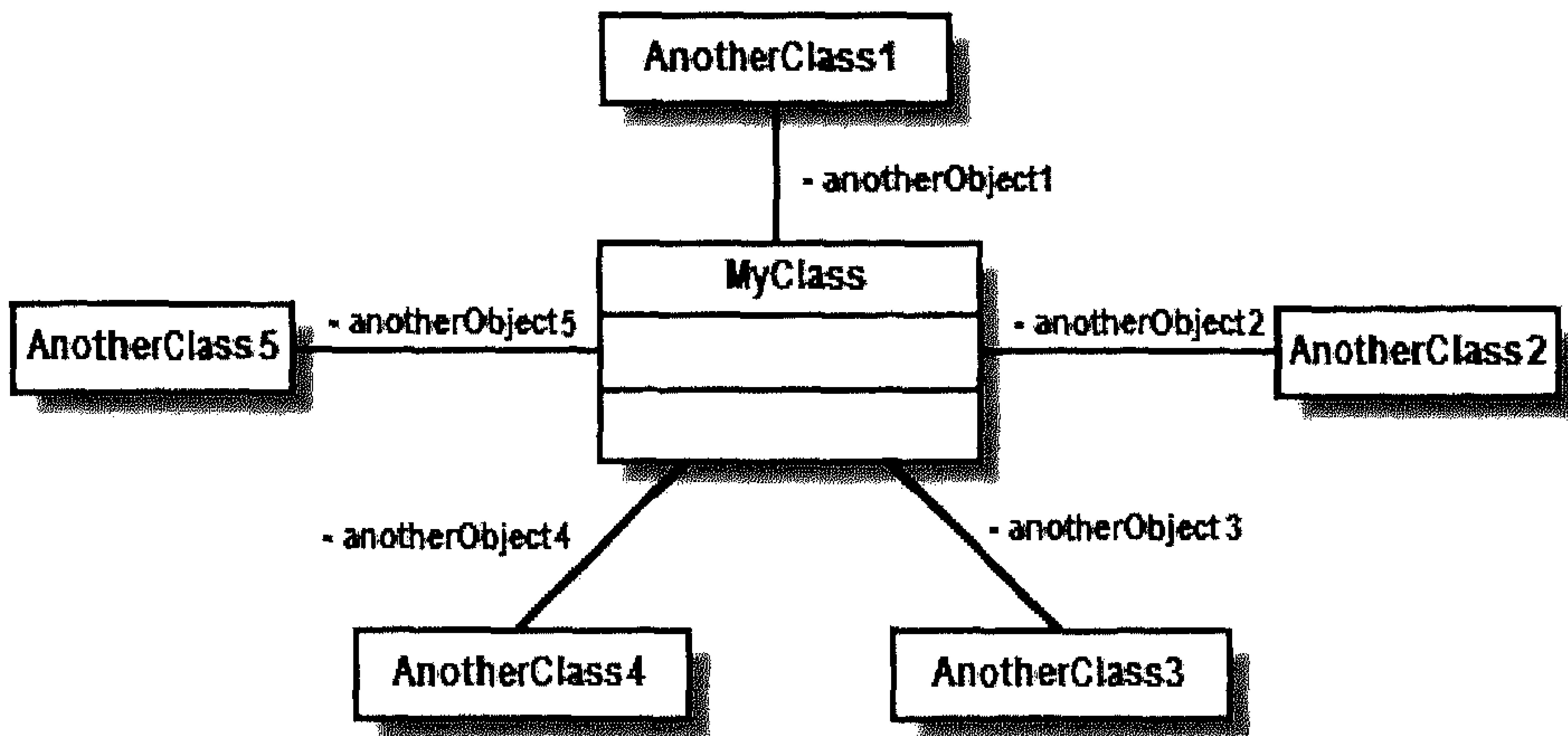


هناك ميزات أخرى تطبق على الخصائص في **UML** تتضمن الميزات التالية: اتحاد **union**، مجموعة فرعية **subsets**، إعادة تعريف **redefines**، وتركيب **composite**. للحصول على وصف دقيق للميزات المختلفة التي يمكن تطبيقها على الخصائص، راجع الكتاب **UML 2.0 in a Nutshell (O'Reilly)**.

٤-٤-٤ الخصائص الضمنية الداخلية Inline مقابل الخصائص بالشراكة

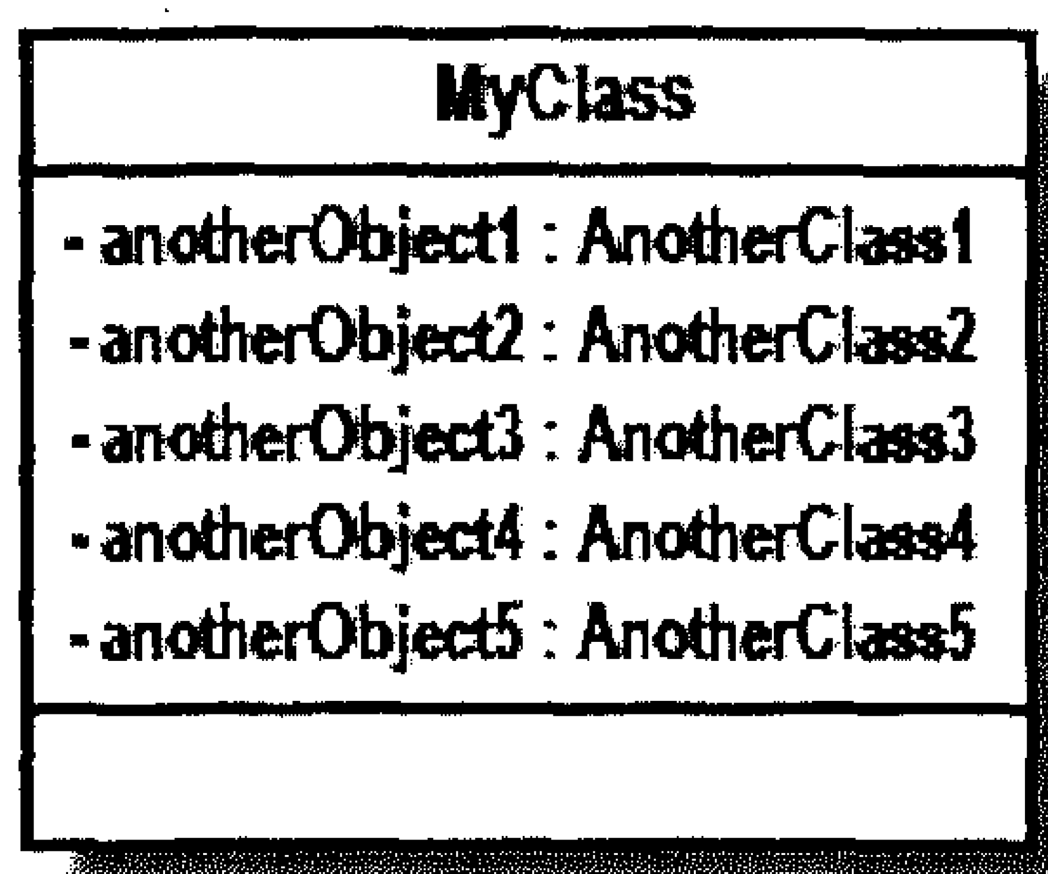
Inline Attributes versus Attributes by Association

لماذا تشويش المسائل باستعمال وسيلتين لعرض خصائص الصنف؟ لاحظ الأصناف والشراكات المعروضة في الشكل رقم (٤-١٤).



شكل رقم (٤-١٤) للصنف MyClass خمسة خصائص معروضة كلها باستعمال الشراكات.

عندما يتم عرض الخصائص على شكل الشراكات، كما هو الحال في الشكل رقم (٤-١٤)، يصبح المخطط مزدحماً بسرعة بمجرد إظهار هذه الشراكات، وذلك دون الاهتمام بكل العلاقات الأخرى بين الأصناف (انظر إلى الفصل الخامس). وعند تحديد الخصائص داخل مستطيل الصنف بأسلوب inline، يصبح المخطط أكثر ترتيباً وأسهل إدارة مع الأقسام الإضافية للمعلومات الأخرى، كما هو معروض في الشكل رقم (٤-١٥).



شكل رقم (٤-١٥) عرض الخصائص الخمسة للصنف **MyClass** بأسلوب **inline** داخل مستطيل الصنف.

إن الاختيار بين إظهار الخاصيّة بأسلوب inline أو من خلال الشراكة هو في الحقيقة سؤال حول ما يركّز عليه المخطط. يبعد استعمال الخصائص inline مركز الاهتمام عن الشراكات التي بين الصنف MyClass والأصناف الأخرى، لكنه يشكل استعمالاً فعّالاً للمكان على المخطط. وتظهر الشركات العلاقات التي بين الأصناف بشكل شديد الوضوح على المخطط، لكنها قد تضايق العلاقات الأخرى كالوراثة التي تكون أكثر أهمية للغاية من مخطط خاص.

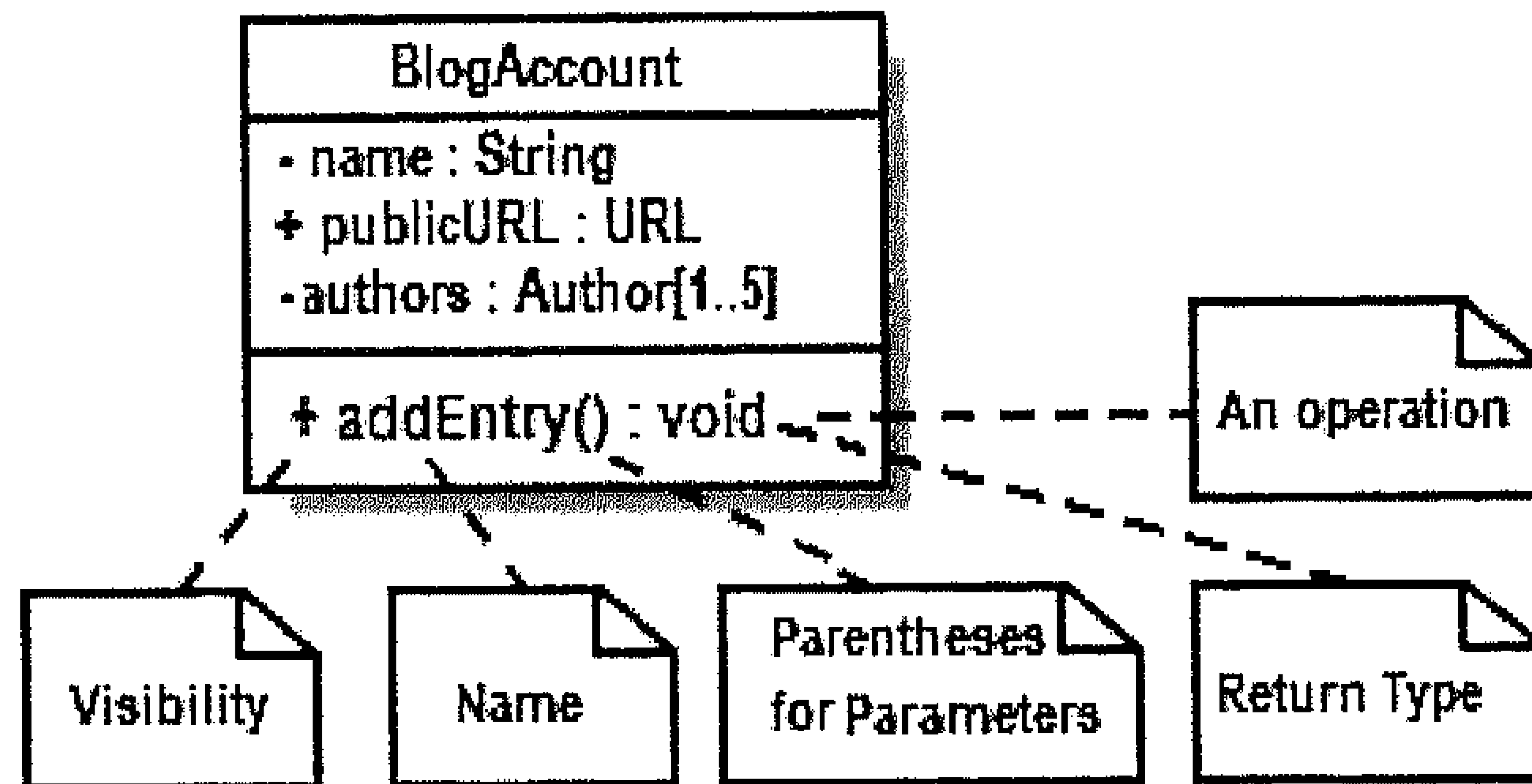


هناك طريقة مجربة مفيدة: تعرض الأصناف "البسيطة" بشكل أفضل باستعمال الأسلوب **inline**، مثل الصنف **String** في جافا، أو حتى أصناف المكتبة البرمجية القياسية كالصنف ملف **File** في الحزمة **io (input output)** بجافا.

٤-٥ سلوكيات الصنف: العمليات
Class Behavior: Operations

تصف عمليات الصنف "ماذا what" يمكن أن يعمله الصنف لكن ليس بالضرورة "كيف how" سيقوم بعمله. تشبه العملية الوعد أو العقد البسيط، الذي يصرح بأن الصنف سيحتوي على بعض السلوكيات التي تعمل ما تقول العملية بأنها ستعمله. ويجب على مجموعة عمليات الصنف أن تشمل كامل السلوكيات التي يحتويها الصنف، بما في ذلك أعمال صيانة خصائص الصنف وربما بعض السلوكيات الإضافية المرتبطة بالصنف.

ويتم تحديد العمليات في UML على مخطط الأصناف بواسطة التوقيع signature، الذي يتألف على الأقل من ميزة الرؤية visibility، واسم name، والأقواس ( ) parentheses لتزويد العملية بأية بارامترات parameters تحتاجها للقيام بعملها، ونوع إرجاع العملية return type، كما هو معروض في الشكل رقم (٤-١٦).

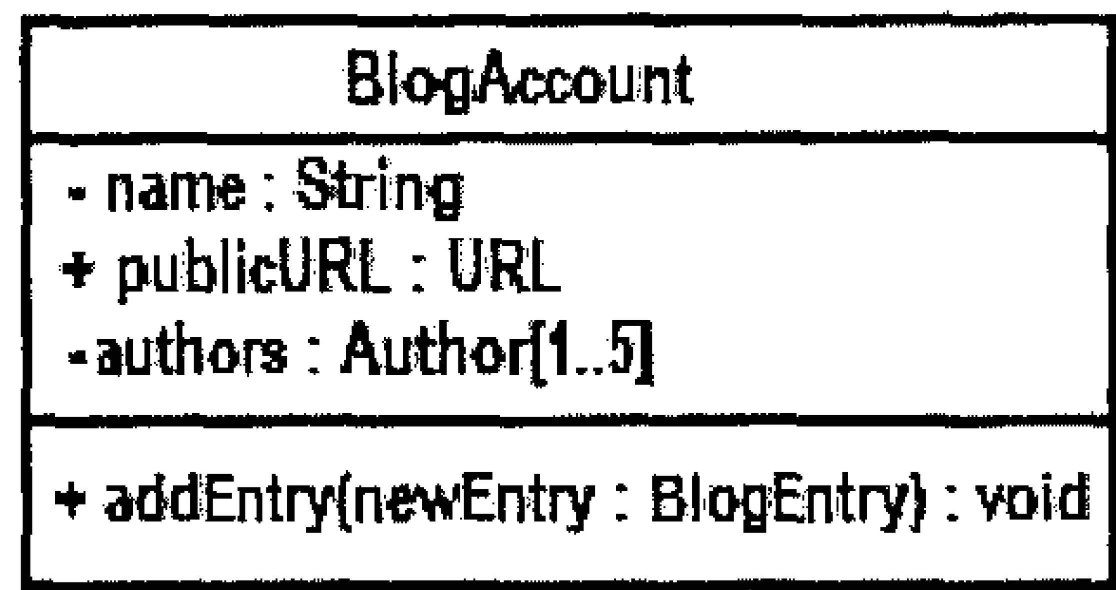


شكل رقم (٤-١٦) إن إضافة عملية جديدة (addEntry) إلى الصنف **BlogAccount** تسمح للأصناف الأخرى بإضافة تدوينات **BlogEntry** إلى المدونة **BlogAccount**.

في الشكل رقم (٤-١٦)، تم التصريح عن العملية **addEntry** باستعمال الرؤية عامة (+)؛ وهي لا تتطلب أي بارامترات لتعمل عليها حتى الآن؛ لأن الأقواس () فارغة، وهي لا ترجع أي قيمة لأن نوع إرجاعها void. وبالرغم من كون هذه العملية صحيحة تماماً في UML، لكنها ليست قريبة حتى من كونها نهائية حتى الآن. يفترض بالعملية إضافة تدوينة BlogEntry جديد إلى المدونة BlogAccount، لكن لا يوجد في الوقت الحاضر وسيلة لمعرفة أية تدوينة يجب إضافتها بالفعل.

٤-٥-١ البارامترات Parameters

تستعمل البارامترات لتحديد المعلومات التي تُزوّد بها العملية للتمكينها من إتمام وظيفتها. على سبيل المثال، تحتاج العملية addEntry(..) إلى تجهيزها بكائن BlogEntry لتتم إضافته إلى حساب المدونة، كما هو معروض في الشكل رقم (٤-١٧).



شكل رقم (٤-١٧) تقلل إضافة بارامتر جديد إلى العملية **addEntry** من الإرباك عند برمجة هذا الصنف؛ على الأقل ستعرف الآن العملية **addEntry** أية تدوينة **BlogEntry** عليها إضافتها إلى المدونة **BlogAccount**.

يشكل البارامتر newEntry الممرر للعملية addEntry في الشكل رقم (٤-١٧) مثالاً بسيطاً لتمرير بارامتر إلى عملية ما. ويجب على الأقل

تحديد النوع البياني للبارامتر، في حالة البارامتر newEntry هو الصنف BlogEntry. ويمكن تمرير أكثر من بارامتر واحد إلى العملية بالفصل بينها باستعمال رمز الفاصلة، كما هو معروض في الشكل رقم (٤-١٨). للمزيد من المعلومات عن الفوارق البسيطة في ترميزات البارامتر، (انظر إلى الكتاب (UML 2.0 in a Nutshell (O'Reilly).

BlogAccount
- name : String + publicURL : URL -authors : Author[1..5]
+ addEntry(newEntry : BlogEntry, author : Author) : void

شكل رقم (٤-١٨) بالإضافة إلى تمرير بارامتر تدوينة جديدة (newEntry) لإضافتها للمدونة، ويمكننا أيضاً تحديد كاتب هذه التدوينة من خلال إضافة البارامتر **author**.

٤-٥-٢ أنواع الإرجاع Return Types

بالإضافة إلى اسم العملية وبارامتراتها، يحتوي توقيع العملية أيضاً على نوع الإرجاع. ويتم تحديد نوع الإرجاع بعد رمز النقطتين العموديتين (:) في نهاية توقيع العملية، والذي يحدد النوع البياني للقيمة التي سترجعها العملية سواء كان نوع بيانات أساسي أو كائن من صنف ما، كما هو معروض في الشكل رقم (٤-١٩).

BlogAccount
- name : String + publicURL : URL -authors : Author[1..5]
+ addEntry(newEntry : BlogEntry, author : Author) : boolean

شكل رقم (٤-١٩) ترجع الآن العملية (..) **addEntry** قيمة منطقية **boolean** تحدد إذا تم إضافة التدوينة بنجاح أم لا.

هناك استثناء واحد لتحديد نوع الإرجاع عند التصرح عن مشيّد الصنف constructor. ويقوم المشيد بإنشاء مثيل جديد من نوع الصنف المُعرّف فيه وإرجاعه كنتيجة له، بناء على ذلك، لا يوجد حاجة إلى تحديد نوع إرجاع للمشيد بشكل صريح، كما هو معروض في الشكل رقم (٤-٢٠).

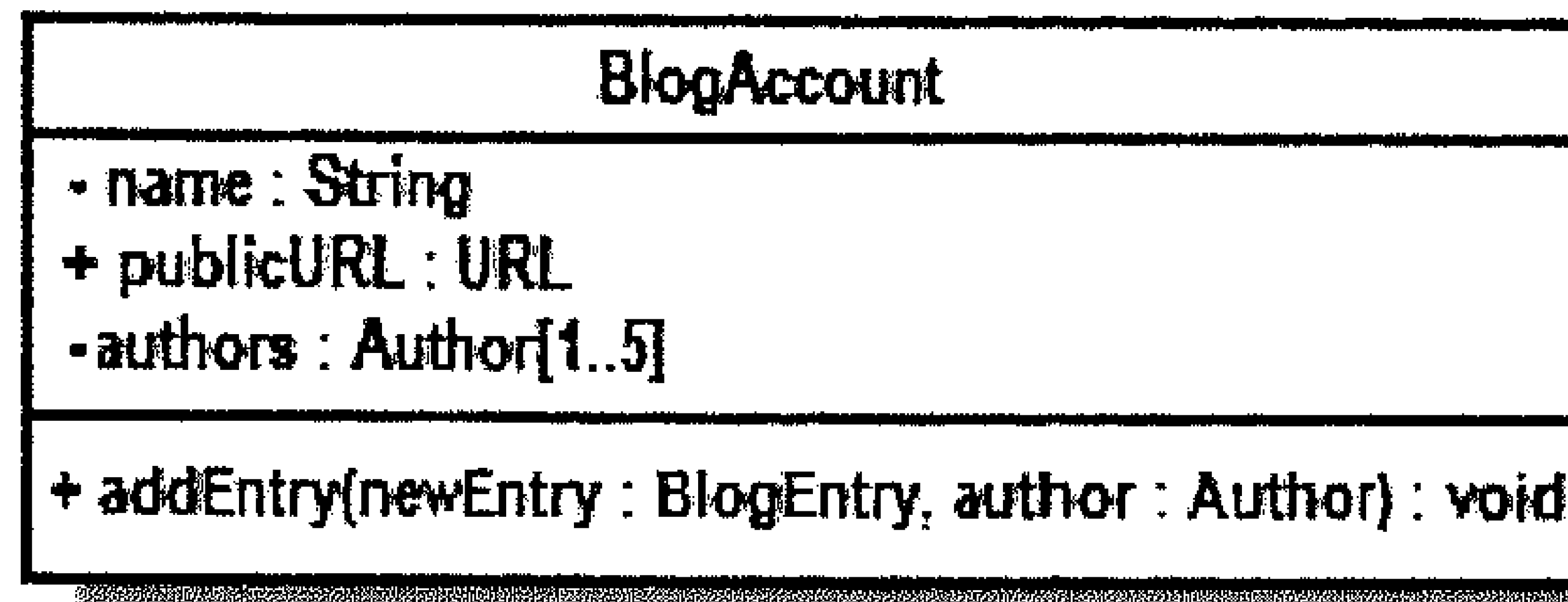
BlogAccount
- name : String + publicURL : URL -authors : Author[1..5]
+ addEntry(newEntry : BlogEntry, author : Author) : boolean + BlogAccount(name : String, publicURL : URL)

شكل رقم (٤-٢٠) يرجع المشيد (..) BlogAccount دائماً مثيلاً من الصنف BlogAccount، لذلك لا يوجد حاجة لإظهار نوع إرجاعه بشكل صريح.

٤-٦ الأجزاء الساكنة من الأصناف
Static Parts of Your Classes

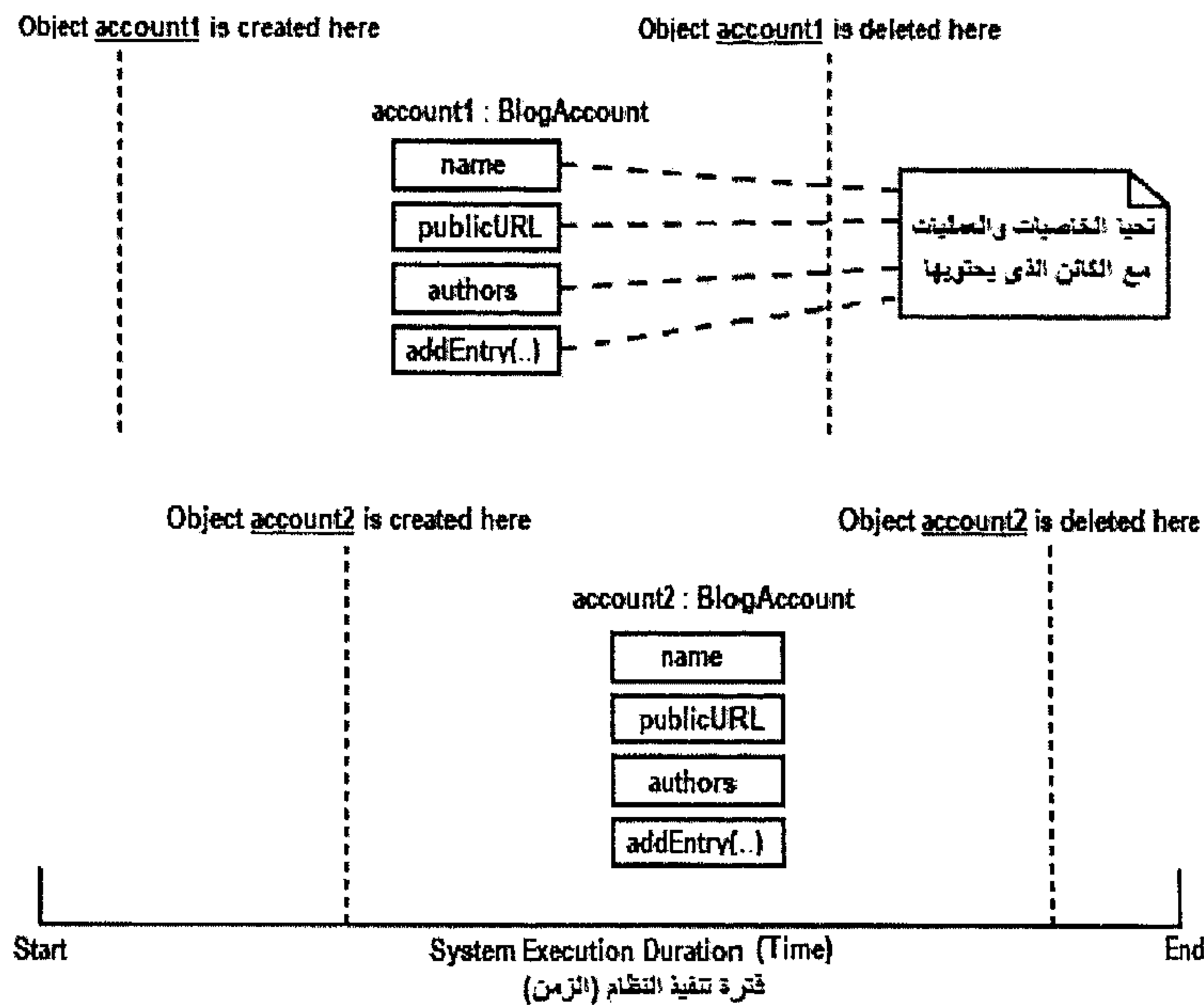
لإنهاء هذه المقدمة عن أساسيات مخططات الأصناف، دعنا نلقي نظرة على إحدى ميزات الأصناف الأكثر إرباكاً: عندما تكون العملية أو الخاصية في الصنف ساكنة static.

يمكن التصريح في UML عن العمليات والخصائص وحتى الأصناف نفسها بأن تكون ساكنة. وللمساعدة على فهم معنى الميزة ساكنة، نحتاج النظر إلى فترة حياة أعضاء الصنف العادية؛ أي غير الساكنة. دعنا أولاً نلقي نظرة أخرى على الصنف BlogAccount المدرج سابقاً في هذا الفصل، انظر إلى الشكل رقم (٤-٢١).



شكل رقم (٤-٢١) يتألف الصنف **BlogAccount** من ثلاث خصائص عادية و من عملية عادية.

وبما أن كل خصائص و عمليات الصنف BlogAccount عادية؛ أي غير ساكنة non-static، فهي ترتبط بمثيلات أو بكائنات الصنف. وهذا يعني أن كل كائن من الصنف BlogAccount سيحصل على نسخة خاصة به من خصائص و عمليات الصنف، كما هو معروض في الشكل رقم (٤-٢٢).



شكل رقم ٤-٢٢. يحتوي ويظهر كلا الحسابين **account1** و **account2** النسخة الخاصة به من الخصائص و العمليات العادية غير الساكنة المصرح عنها في الصنف **BlogAccount**.

تريد أحياناً من كل كائنات الصنف مشاركة نفس النسخة من خاصية أو عملية ما. وعند حدوث ذلك، يتم ربط خصائص وعمليات الصنف بالصنف نفسه، ويكون لها فترة حياة تستمر أبعد من حياة أي كائن منشأ من الصنف. من هنا تصبح الخصائص و العمليات الساكنة مفيدة.

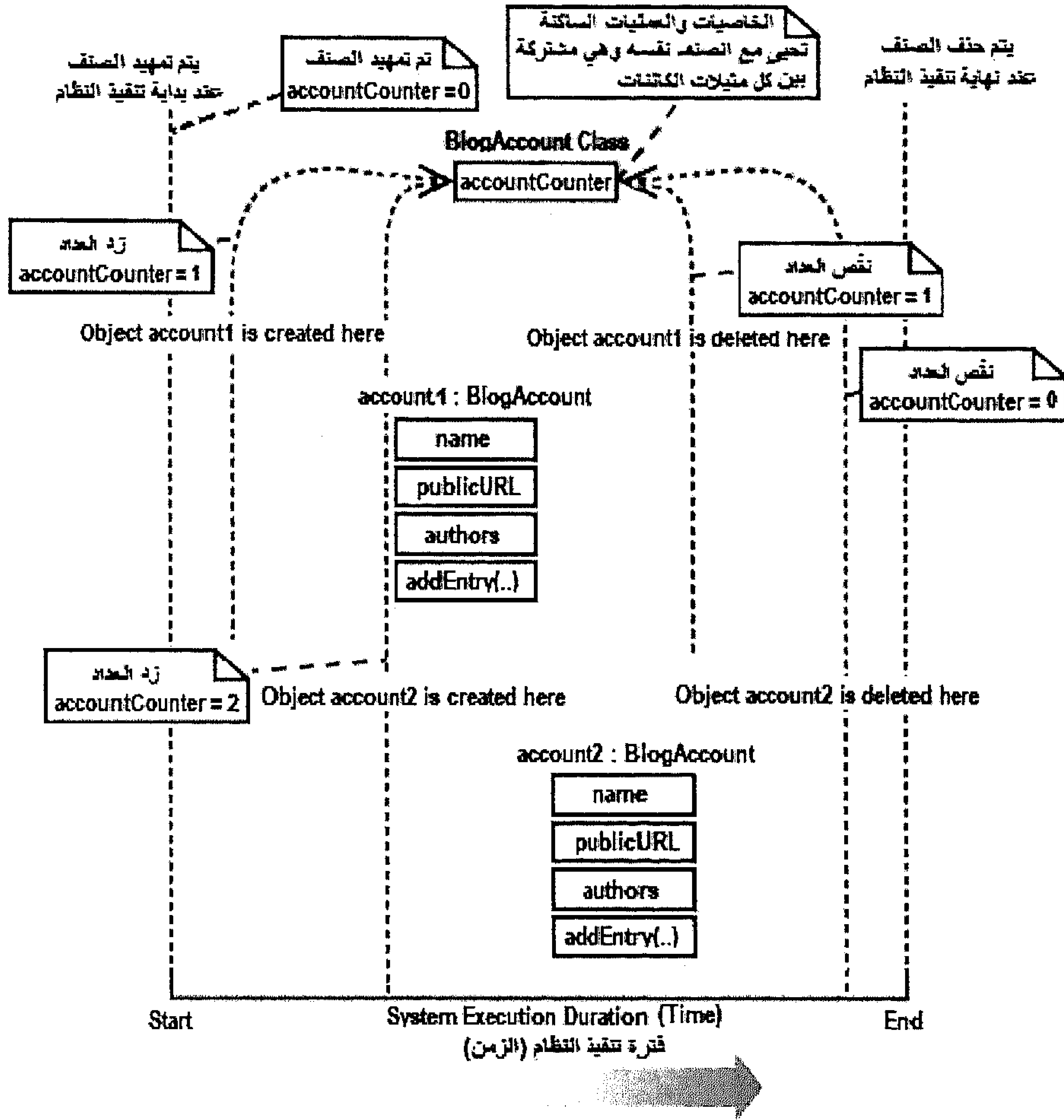
على سبيل المثال (دعنا نتجاهل هنا احتمال ملقّمات الصنف المتعددة class-loaders)، إذا أردنا الاحتفاظ بعداد لكل كائنات BlogAccount النشطة في النظام، ويشكل حينئذ هذا العداد مرشّحاً جيداً ليكون خاصية ساكنة للصنف. بدلاً من ربط خاصية العداد بأي كائن من الصنف، فيتم ربطها بالصنف BlogAccount نفسه، وتكون بالتالي خاصية ساكنة كما هو معروض في الشكل رقم (٤-٢٣).

BlogAccount
- name : String + publicURL : URL -authors : Author[1..5] -accountCounter : int
+ addEntry(newEntry : BlogEntry, author : Author) : void

شكل رقم (٤-٢٣) يوضع تحت الخاصية أو العملية خطاً للتحديد بأنها ساكنة في UML؛ يتم استعمال الخاصية الساكنة accountCounter للاحتفاظ بعداد واحد لعد الكائنات النشطة من الصنف BlogAccount.

تحتاج الخاصية عداد الحسابات **accountCounter** إلى زيادتها عند إنشاء أي كائن جديد من الصنف BlogAccount. ولقد تم التصريح عن الخاصية accountCounter بأنها ساكنة بسبب الحاجة إلى مشاركة

نسخة واحدة منها بين كل كائنات الصنف BlogAccount. ويمكن أن تقوم الكائنات بزيادتها عند إنشائها و أن تقوم بتنقيصها عند تدميرها، كما هو معروض في الشكل رقم (٤-٢٤).



شكل رقم (٤-٢٤) تكون الخاصية الساكنة accountController مشتركة بين مختلف كائنات BlogAccount للاحتفاظ بعدد كائنات BlogAccount النشطة داخل النظام.

إذا كانت الخاصية accountCounter غير ساكنة، يحصل بالتالي كل كائن BlogAccount على نسخة خاصة به من الخاصية accountCounter. وهذا لن يفيد إطلاقاً؛ لأن كل كائن BlogAccount سيقوم فقط بتحديث النسخة الخاصة به من accountCounter بدلاً من المشاركة في عداد رئيسي للكائنات. وفي الحقيقة، إذا لم تكن accountCounter ساكنة، سيقوم كل كائن ببساطة بزيادة النسخة الخاصة به إلى القيمة ١ عند إنشائه وبعد ذلك بتنقيصها إلى القيمة ٠ عند تدميره، وهذا غير مفيد على الإطلاق!

نمط تصميم المثيل الوحيد The Singleton Design Pattern

هناك مثال آخر مهم لاستعمال الخصائص والعمليات الساكنة، وذلك عند تطبيق نمط التصميم المثيل الوحيد. باختصار، يضمن نمط تصميم المثيل الوحيد إنشاء كائن واحدٍ وواحد فقط من نوع صنف ما أثناء فترة حياة النظام. ولضمان إنشاء كائن واحد فقط، تحتفظ البرمجة النموذجية لنمط تصميم المثيل الوحيد بمرجع ساكن داخلي يشير إلى الكائن الوحيد المسموح به، ويتم التحكم بالوصول إلى هذا الكائن باستعمال عملية ساكنة. (راجع الكتاب Head First Design Patterns (O'Reilly) لتتعلم أكثر عن نمط تصميم المثيل الوحيد).

٤-٧ ما هي الخطوة التالية؟

لقد قدم لنا هذا الفصل نظرة أولية عن كل ما هو جائز مع مخططات الأصناف. ويمكن أن يوجد علاقات بين الأصناف، كما يُوْجِدُ حتى أشكالاً متقدمة من الأصناف، مثل القوالب templates، التي تجعل تصميم النظام أكثر فعّالية. وستتم تغطية علاقات الأصناف و الأصناف المجردة abstract و القوالب في الفصل الخامس.

ويظهر مخطط الأصناف أنواع الكائنات التي في النظام؛ ويمكن النظر في مخططات الكائنات كخطوة قادمة مفيدة، لأنها تظهر كيف تصبح الأصناف نشطة وقت التشغيل على شكل كائنات مثيلة لها، والتي تكون مفيدة في إظهار ترتيبات وقت التشغيل. وستتم تغطية مخططات الكائنات في الفصل السادس.

إن الهياكل المركبة composite structures هي نوع من المخططات التي تعرض بحرية تامة مخططات الأصناف المتأثرة بالسياق context-sensitive والأنماط patterns في البرنامج. وسيتم وصف الهياكل المركبة في الفصل الحادي عشر.

بعد القيام بتحديد مسؤوليات الأصناف التي في النظام، من الشائع إنشاء مخططات التتابع والاتصال sequence and communication diagrams التي تعرض التفاعلات بين الأجزاء. يمكن أن تجد مخططات التتابع في الفصل السابع. وستتم تغطية مخططات الاتصال في الفصل الثامن.

من الشائع أيضاً الرجوع للوراء و تنظيم الأصناف في حُزَم packages. وتسمح لك مخططات الحُزَم برؤية التبعيّات من مستوى أعلى، وتساعد على فهم استقرار البرنامج. وسيتم وصف مخططات الحزم في الفصل الثالث عشر.

نمذجة الهيكل المنطقي للنظام: مخططات الأصناف المتقدمة

MODELING A SYSTEM'S LOGICAL STRUCTURE: ADVANCED CLASS DIAGRAMS

إذا كانت مخططات الأصناف تسمح فقط بالتصريح عن أصناف لديها خصائص وعمليات بسيطة، فإن لغة النمذجة الموحدة بالتالي تكون لغة نمذجة هزيلة جداً. لحسن الحظ، يسمح التوجه الكائني ولغة النمذجة الموحدة بعمل أمور أكثر بكثير مع الأصناف من مجرد التصريحات البسيطة فقط. فبالنسبة للمبتدئين، يمكن أن يكون للأصناف علاقات مع بعضها بعضا. ويمكن للصنف أن يكون من نوع صنف آخر (من خلال التعميم) أو يمكن أن يحتوي على كائنات صنف آخر بأساليب مختلفة بالاعتماد على قوة العلاقة بينهما.

تفيد الأصناف المجردة في التصريح جزئياً عن سلوك الصنف، ومما يسمح لأصناف أخرى بتكملة الأجزاء الناقصة من السلوك – المجردة – بالطريقة التي تناسبها. تأخذنا الواجهات interfaces درجة أبعد من الأصناف المجردة في التجرد من خلال توصيفها للعمليات الضرورية للصنف فقط من دون إنجاز أي منها. ويمكن أيضاً تطبيق القيود على مخططات

الأصناف باستعمال لغة قيود الكائن Object Constraint Language (OCL) التي تصف كيفية استعمال كائنات الصنف.

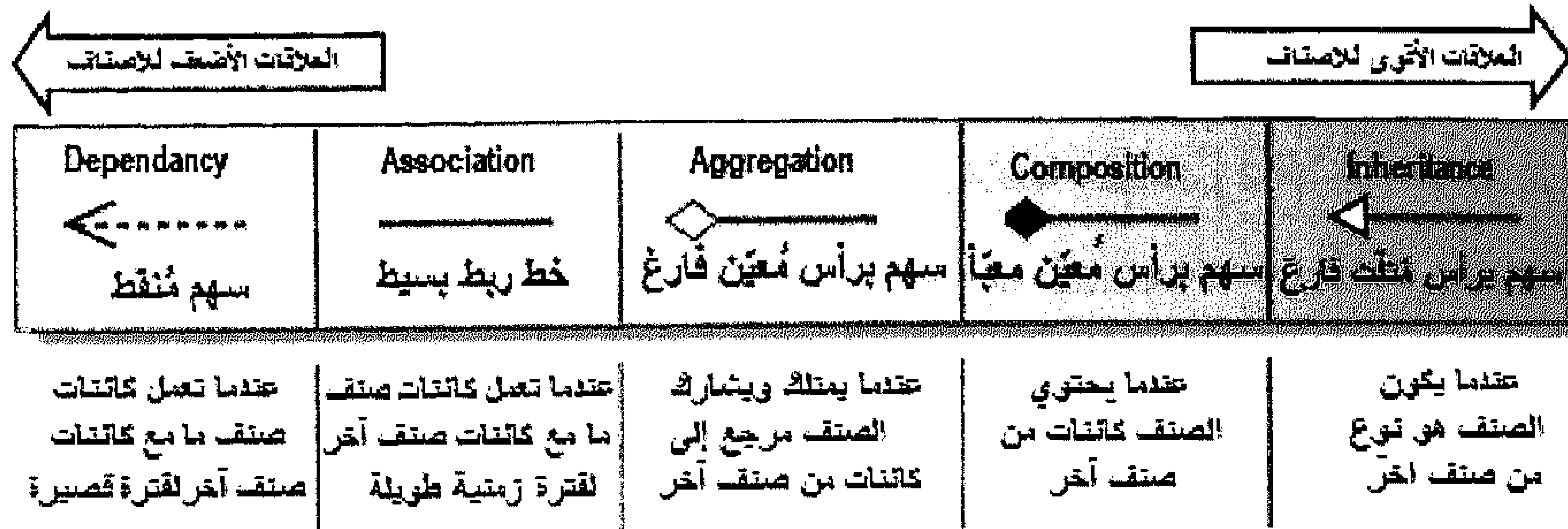
ويكتمل الوصف مع القوالب templates التي تسمح بالتصريح عن أصناف تحتوي على سلوك عام وقابل للاستعمال ثانية كلياً. ويمكن مع القوالب تحديد ما سيعمله الصنف القالب ثم الانتظار – إذا أردت حتى وقت التشغيل – لتقرّر مع أي أصناف سيعمل هذا الصنف القالب.

وتقوم كل هذه التقنيات بتكملة صندوق أدوات مخطط الأصناف. وهي تمثل عينة من أقوى المفاهيم في التصميم الكائني التوجه، كما أنها تشكل الفرق بين التصميم المقبول وجزء عظيم من التصميم القابل للاستعمال ثانية عند تطبيقها بشكل صحيح.

٥-١ علاقات الصنف Class Relationships

لا تعيش الأصناف في الفراغ، بل تعمل معاً باستعمال أنواع مختلفة من العلاقات فيما بينها. ويكون لهذه العلاقات قوى مختلفة، كما هو معروض في الشكل رقم (٥-١).

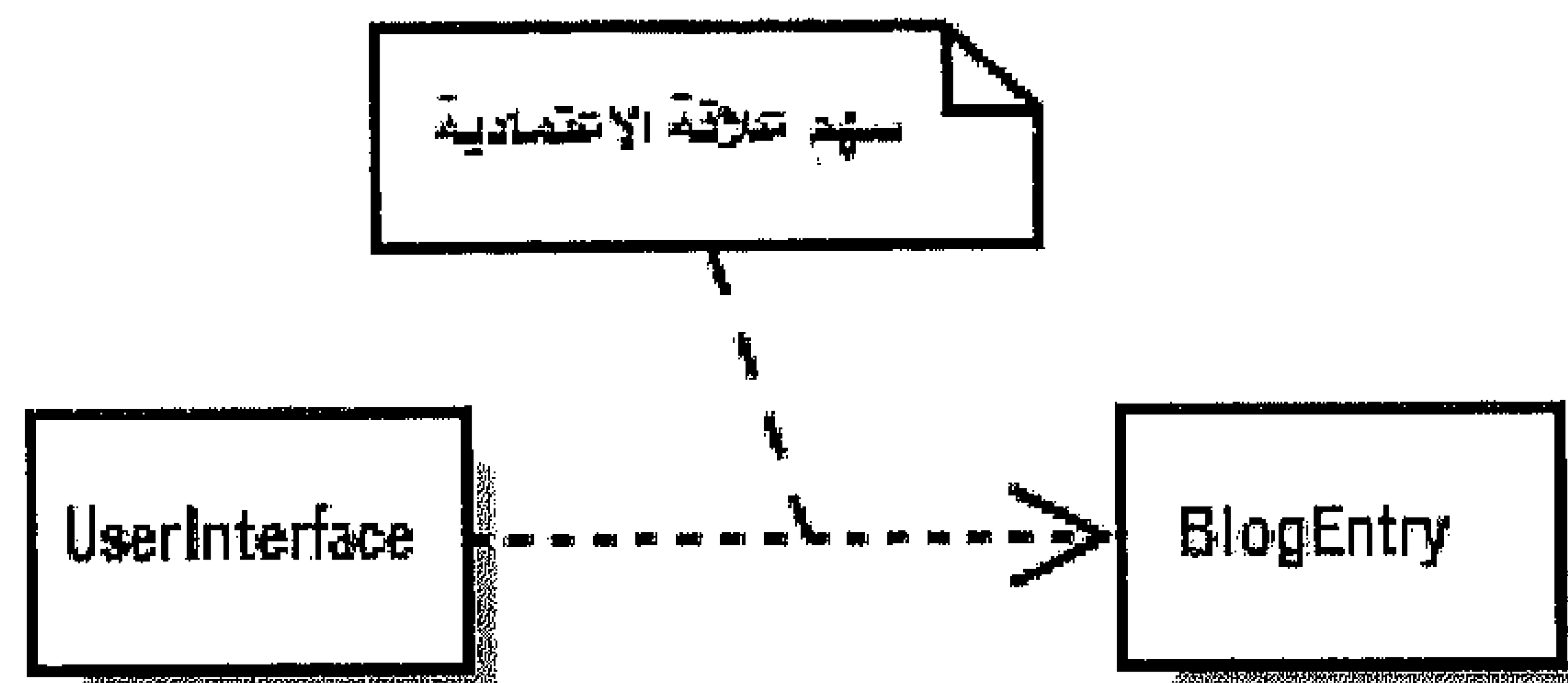
ترتكز قوة العلاقة بين الأصناف على مدى اعتماد الأصناف المشاركة في هذه العلاقة على بعضها بعضا. إن الصنفين المعتمد أحدهما بقوة على الآخر يقال أنهما مقترنان بإحكام **tightly coupled**؛ وغالباً ما تؤثر التغييرات في أحد الصنفين على الصنف الآخر. وعادة ما يكون الاقتران المحكم أمراً سيئاً لكن ليس دائماً؛ لذلك، بقدر ما تكون العلاقات أقوى، بقدر ما نحتاج إلى الحذر و الاحتراس.



شكل رقم (٥-١) تقدم UML خمسة أنواع مختلفة من العلاقات بين الأصناف.

٥-١-١ التبعية Dependency

تصرح علاقة التبعية بين صنفين أن صنفاً ما يحتاج إلى أن يعرف بشأن صنف آخر لاستعمال كائناته. إذا احتاج الصنف UserInterface من نظام إدارة المحتوى CMS إلى العمل مع كائن من الصنف BlogEntry، يتم بالتالي رسم هذه التبعية باستعمال سهم التبعية، كما هو معروض في الشكل رقم (٥-٢).



شكل رقم (٥-٢) يعتمد الصنف **UserInterface** على الصنف **BlogEntry**؛ لأنه يحتاج إلى قراءة محتوى التدوينات لعرضها على المستخدم.

يعمل الصنفان UserInterface و BlogEntry معاً عندما تريد واجهة المستخدم عرض محتوى التدوينات. وفي مصطلحات مخطط الأصناف، يعتمد صنفاً الكائن على بعضهما بعضا لضمان عملهما معاً في وقت التشغيل.

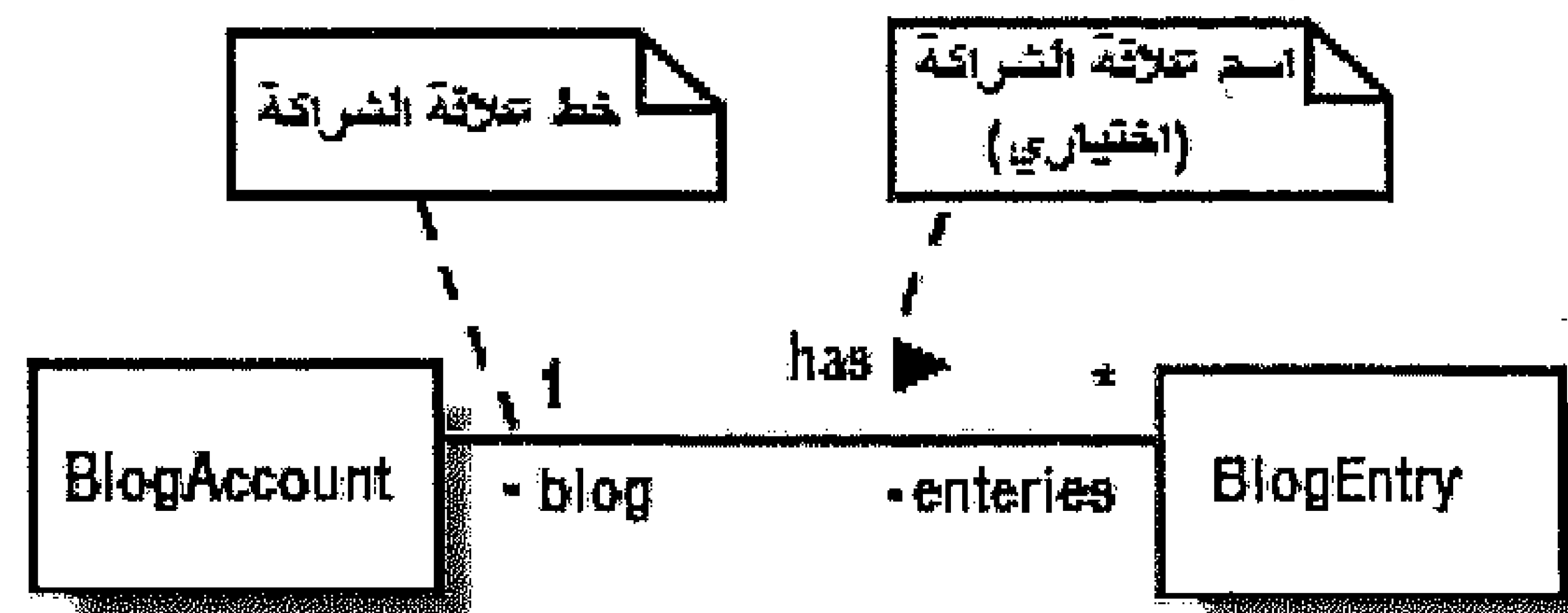
وتدل التبعية فقط على أن كائنات الصنف يمكن أن تعمل معاً؛ بناء على ذلك، تعتبر علاقة التبعية أنها العلاقة المباشرة الأضعف التي يمكن أن نجدها بين صنفين.



عادة ما تستعمل علاقة التبعية عندما يكون صنف ما يوفر مجموعة دوال مفيدة وعامة الأهداف، مثل حُزَم جافا الخاصة بالتعابير المنتظمة regular expression (java.util.regex) أو بالرياضيات (java.math). وتعتمد الأصناف الجديدة على الأصناف الموجودة في java.util.regex و java.math لاستعمال الخدمات التي توفرها.

٥-١-٢ الشراكة Association

بالرغم من أن علاقة التبعية تسمح لصنف ما باستعمال كائنات صنف آخر، فإن علاقة الشراكة تعني أن صنفاً محدداً سيحتوي فعلياً على مرجع إلى كائن أو كائنات من الصنف الآخر على شكل خاصية له. إذا وجدت نفسك تقول أن صنفاً محدداً يعمل مع كائن من صنف آخر، تكون بالتالي العلاقة التي بين تلك الأصناف مرشحة بدرجة كبيرة لتكون علاقة شراكة بدلاً من مجرد علاقة تبعية. ويتم عرض علاقة الشراكة باستعمال خط ربط بسيط يربط بين الصنفين، كما هو معروض في الشكل رقم (٥-٣).



شكل رقم (٥-٣) يشترك الصنف BlogAccount بشكل اختياري مع صفر أو أكثر من كائنات الصنف BlogEntry؛ يشترك أيضاً BlogEntry مع كائن واحد من BlogAccount.

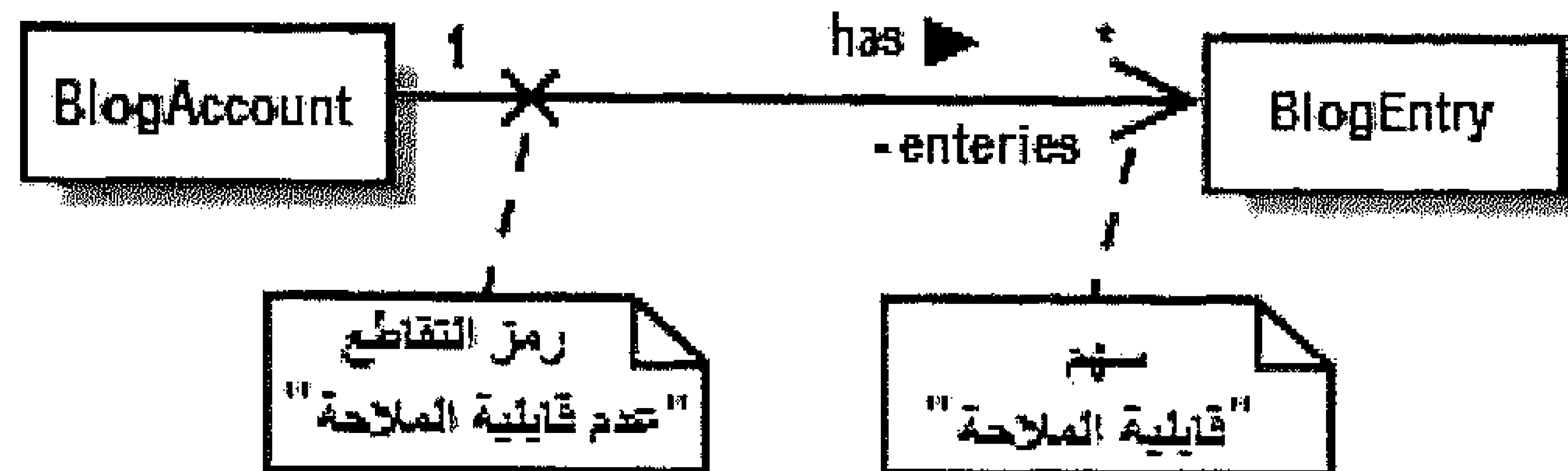
عادة ما تطبق قابلية الملاحة **navigability** على علاقة الشراكة لوصف الصنف الذي يحتوي على الخاصية الداعمة للعلاقة. إذا أخذت الشكل رقم (٥-٣) كما هو حالياً و أنجزت بلغة جافا الشراكة التي بين الصنفين، ستحصل بالتالي على شيء يشبه ما هو معروض في المثال رقم (٥-١).

مثال رقم (٥- ١) عرض الصنفين **BlogAccount** و **BlogEntry** من دون تطبيق قابلية الملاحة على علاقة الشراكة التي بينهما.

```
public class BlogAccount {
   // BlogEntry الخاصية المدخلة بفضل الشراكة مع الصنف
   private BlogEntry[] entries;
   //... يصرح عن الخصائص و الطرق الأخرى هنا...
}
public class BlogEntry {
   // Blog الخاصية المدخلة بفضل الشراكة مع الصنف
   private BlogAccount blog;
   //... يصرح عن الخصائص و الطرق الأخرى هنا...
}
```

من دون أي معلومات إضافية عن الشراكة بين الصنفين BlogAccount و BlogEntry، من المستحيل تقرير أي صنف يجب أن يحتوي على الخاصية المدخلة من قبل الشراكة؛ وفي هذه الحالة، يتم إضافة خاصية في كلا الصنفين. ربما لا يوجد مشكلة إذا تعمدنا ذلك؛ لكن، من الشائع وجود صنف واحد فقط يشير إلى الصنف الآخر في الشراكة. مثلاً في نظام إدارة المحتوى، إذا استطعنا سؤال حساب مدونة عن التدوينات التي يحتويها، يكون لذلك معنى أكثر من سؤال التدوينة عن حساب المدونة الذي تنتمي إليه. وفي هذه الحالة، نستعمل قابلية الملاحة

لضمان حصول الصنف BlogAccount على الخاصية المدخلة من قبل الشراكة، كما هو معروض في الشكل رقم (٥-٤).



شكل رقم (٥-٤) إذا غيرنا الشكل رقم (٥- ٣) ليحتوي سهم قابلية الملاحة، يمكننا بالتالي تحديد أنه يجب أن نكون قادرين على الإبحار من المدونة إلى تدويناتها.

تؤدي عملية تحديث علاقة الشراكة بين الصنف BlogAccount والصنف BlogEntry المعروضة في الشكل رقم (٥-٤)، إلى شفرة البرمجة التي في المثال رقم (٥-٢).

مثال رقم (٥-٢) مع تطبيق قابلية الملاحة، يحتوي الصنف **BlogAccount** فقط على الخاصية المدخلة من قبل الشراكة.

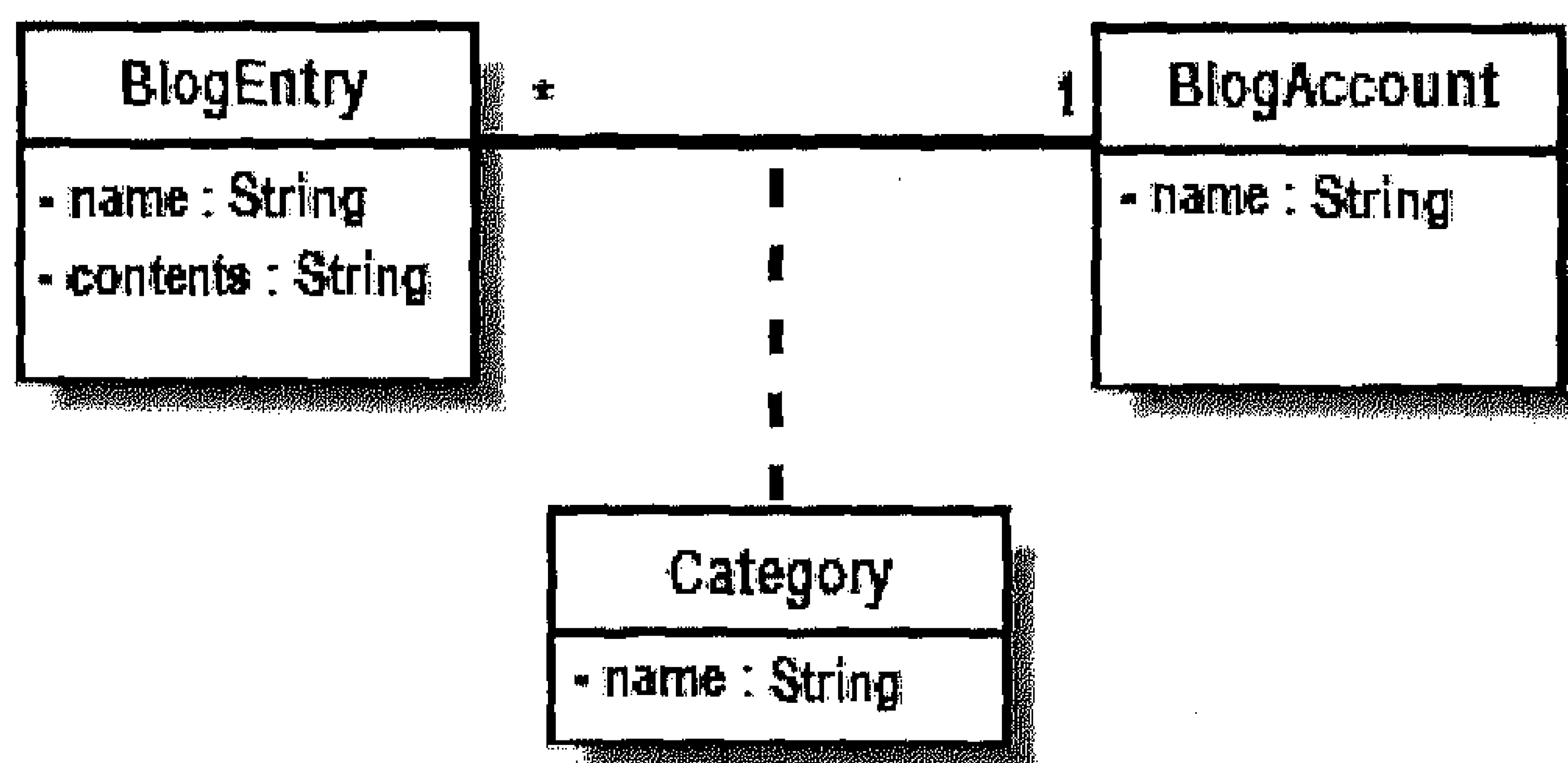
```
public class BlogAccount {
   // BlogEntry الخاصية المدخلة بفضل الشراكة مع الصنف
   private BlogEntry[] entries;
   //... يصرح عن الخصائص و الطرق الأخرى هنا...
}
public class BlogEntry {
   // BlogEntry لقد تم حذف خاصية حساب المدونة بما أنها غير ضرورية للصنف
   // ليعرف عن حساب المدونة الذي ينتمي
   //... يصرح عن الخصائص و الطرق الأخرى هنا...
}
```

٥-١-٢-١ أصناف الشراكة Association Classes

تقوم أحياناً علاقة الشراكة نفسها بإدخال أصناف جديدة. تفيد أصناف الشراكة بشكل خاص مع الحالات المعقدة عندما تريد عرض أن صنفاً ما مرتبط بصنفَين؛ لأنهما مرتبطان بعلاقة فيما بينهما، كما هو معروض في الشكل رقم (٥-٥).

وفي الشكل رقم (٥-٥)، يوجد علاقة شراكة بين الصنفَين BlogEntry و BlogAccount. بالاعتماد أيضاً على الفئات categories التي يحتويها الحساب، تشترك أيضاً التدوينة مع أي عدد من الفئات. باختصار، تنتج علاقة الشراكة التي بين حساب مدونة BlogAccount وتدوينة BlogEntry علاقة شراكة أخرى مع مجموعة من الفئات.

لا توجد قواعد صارمة و سريعة لكيفية برمجة صنف الشراكة بالضبط، لكن يمكن برمجة العلاقات المعروضة في الشكل رقم (٥-٥) بلغة جافا (على سبيل المثال) كما هو معروض في المثال رقم (٥-٣).



شكل رقم (٥-٥) يشترك BlogEntry مع Category بسبب اشتراكه مع BlogAccount محدد.

مثال رقم (٥-٣) يعرض طريقة لبرمجة علاقة الصنف **BlogEntry** بالصنف **BlogAccount** و صنف الشراكة **Category** بلغة جافا.

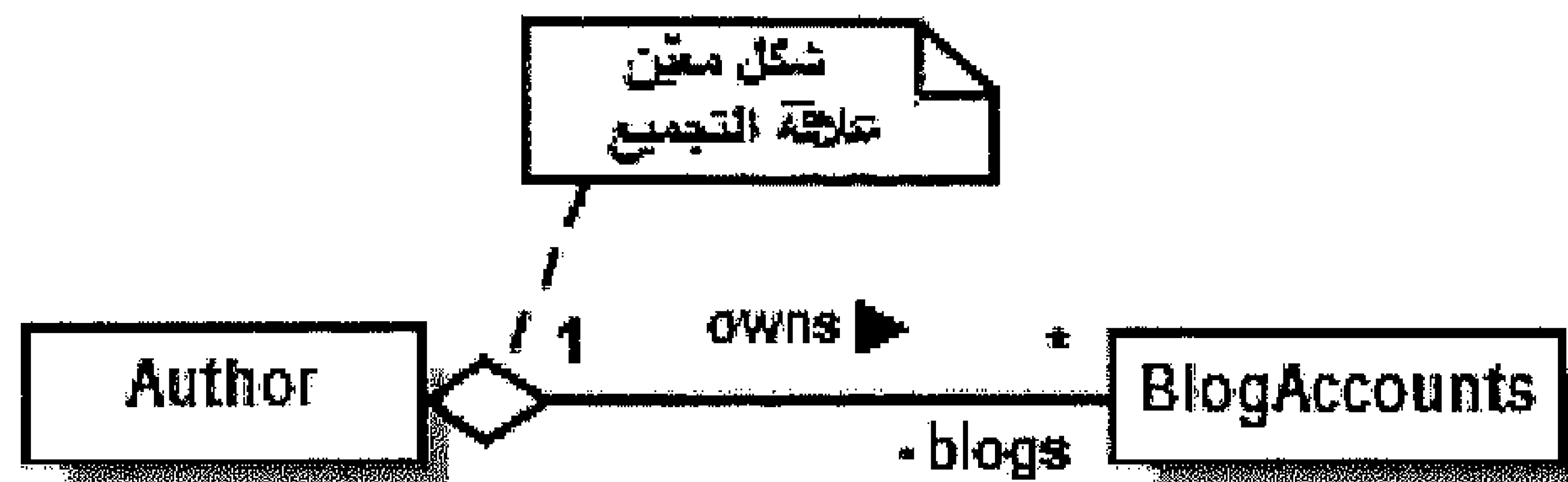
```
public class BlogAccount {
   private String name;
   private Category[] categories;
   private BlogEntry[] entries;
}

public class Category {
    private String name;
}

public class BlogEntry {
   private String name;
   private String contents;
   private Category[] categories;
}
```

٥-١-٣ علاقة التجميع Aggregation

بالتقدم خطوة إضافية إلى الأمام انطلاقاً من علاقة الشراكة نجد علاقة التجميع. وتشكل علاقة التجميع في الحقيقة مجرد نسخة أقوى لعلاقة الشراكة، ويتم استعمالها للإشارة إلى امتلاك وربما مشاركة الصنف كائنات من صنف آخر. ويتم عرض علاقة التجميع باستعمال رأس سهم ذات شكل معيّن فارغ من جانب الصنف المالك، كما هو معروض في الشكل رقم (٥-٦).



شكل رقم (٥-٦) تظهر علاقة التجميع أن كاتباً ما **Author** يمتلك مجموعة حسابات مدونة **blogs** من الصنف **BlogAccounts**.

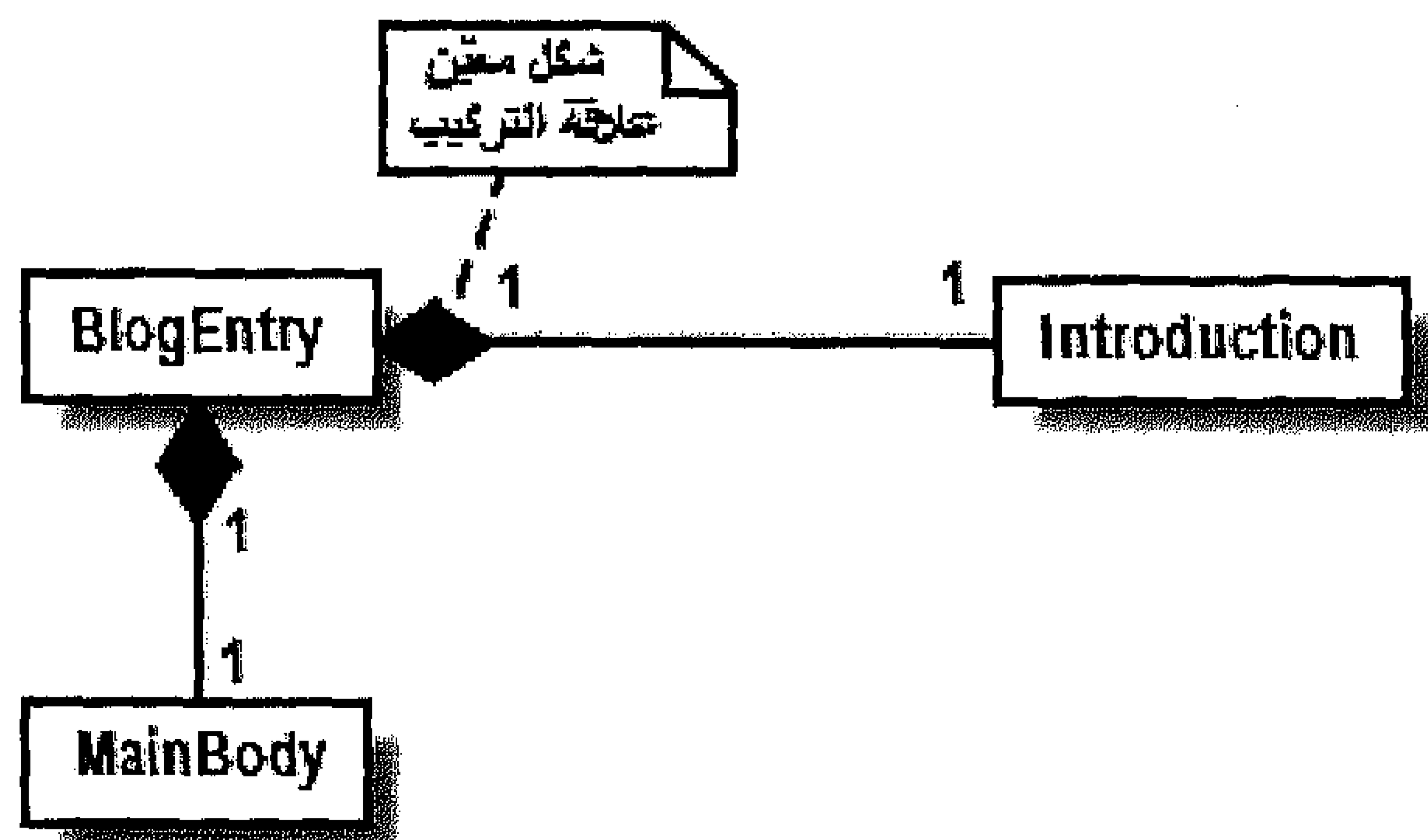
إن العلاقة بين كاتب وحساباته بالمدونة، كما هو معروض في الشكل رقم (٥-٦)، هي أقوى بكثير من مجرد علاقة الشراكة. ويمتلك الكاتب حساباته التي بالمدونة، رغم أنه قد يشاركهم مع كُتَّاب آخرين، إلا أنه تبقى تلك الحسابات بالنهاية ملكه، وإذا قرر أن يحذف أحدها فيمكنه ذلك.



أين شفرة برمجة علاقة التجميع؟ في الحقيقة، شفرة جافا لبرمجية علاقة التجميع هي نفس شفرة برمجة علاقة الشراكة بالضبط؛ فهي تؤدي إلى إدخال خاصية ما.

٥-١-٤ علاقة التركيب Composition

بالتقدم خطوة إضافية على خط علاقات الصنف، نجد أن علاقة التركيب هي علاقة أقوى أيضاً من علاقة التجميع، رغم أنهما تعملان بأساليب متشابهة جداً. ويتم عرض علاقة التركيب باستعمال رأس سهم ذات شكل معيّن معبّأ، كما هو معروض في الشكل رقم (٥-٧).



شكل رقم (٥-٧) تتألف التدوينة **BlogEntry** من مقدمة **Introduction** وجسم أساسي **MainBody**.

يشكل قسم المقدمة وقسم الجسم الأساسي للتدوينة أجزاء فعلية من التدوينة نفسها، وعلى الأغلب لن تتم مشاركتهما مع أجزاء أخرى من النظام. وإذا تم حذف التدوينة، فيتم بالتالي حذف أجزائها معها أيضاً. هذه حقيقة علاقة التركيب بالضبط. نمذجة الأجزاء الداخلية المؤلفة للصنف.

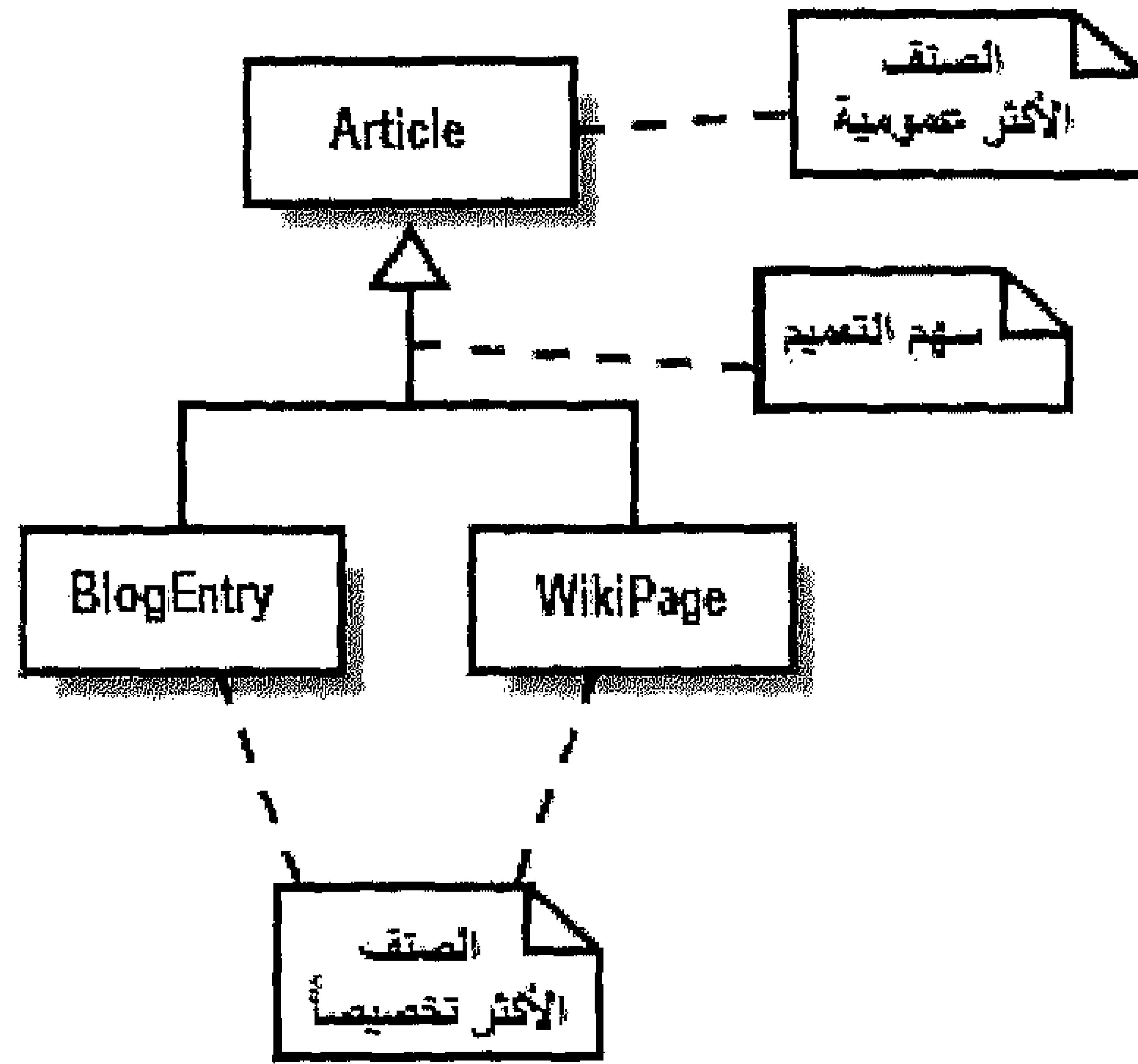


> بشكل مشابه لعلاقة التجميع، تؤدي شفرة برمجة علاقة التركيب بلغة جافا إلى إدخال خاصية ما فقط.

٥-١-٥ التعميم Generalization (أو الوراثة Inheritance)

تستعمل علاقة التعميم أو الوراثة لوصف الصنف بأنه "هو نوع من is a type of" صنف آخر. ولقد أصبح التعبير "له has a" والتعبير " هو نوع من is a type of"، منذ عدة سنوات، وسيلة مقبولة لتحديد إذا كانت العلاقة التي بين الصنفين هي علاقة تجميع أو علاقة تعميم. إذا وجدت نفسك تقول بأن الصنف له جزء عبارة عن كائن من صنف آخر، يرجّح بالتالي أن تكون العلاقة التي بينهما هي علاقة "شراكة" أو "تجميع" أو "تركيب". لكن إذا وجدت نفسك تقول بأن الصنف هو من نوع صنف آخر، عندئذ تكون العلاقة التي بينهما هي علاقة "تعميم" بدلاً من تلك العلاقات.

في لغة النمذجة الموحدة، يتم استعمال سهم التعميم (مثلث فارغ) لعرض أن الصنف هو من نوع صنف آخر، كما هو معروض في الشكل رقم (٥-٨).



شكل رقم (٥-٨) يعرض أن التدوينة **BlogEntry** و صفحة الويكي **WikiPage** كلاهما من النوع مقالة **Article**.

إن الصنف الأكثر تعميماً الذي تتم وراثته في علاقة التعميم موجود عند نهاية السهم (الصنف مقالة Article في الشكل رقم ٥-٨)، وغالباً ما يشار إليه مثل الصنف الأهل parent أو الصنف الأساس base أو الصنف العلوي superclass. إن الأصناف الأكثر تخصيصاً هي التي ترث، مثل الصنفَين BlogEntry و WikiPage في الشكل رقم (٥-٨)، يشار إليها غالباً مثل الأصناف الأبناء children أو المشتقة derived. ويرث الصنف المخصّص كل الخصائص والطرق المصرح عنها في الصنف المعمّم، وربما يضيف عمليات وخصائص تكون قابلة للتطبيق فقط في الحالات المخصّصة.

تأتي تسمية الوراثة بالتعميم في لغة النمذجة الموحدة من الاختلاف بين ما يمثله الصنف الأهل والصنف الابن. وتصف الأصناف الأهل الأنواع الأكثر عمومية، بالمقابل تصف الأصناف الابن الأنواع الأكثر تخصّصاً.



إذا احتجت التحقق من صحة علاقة التعميم، تفيد هذه الطريقة المجربة: يكون لعلاقة التعميم معنى في اتجاه واحد فقط. مع أنه يصح القول أن عازف القيثارة هو موسيقي، لكن لا يصح القول أن كل الموسيقيين هم عازفو قيثارة.

٥-١-٥-١ التعميم و إعادة استعمال الشفرة

Generalization and implementation reuse

يقوم الصنف الابن بوارثة وإعادة استعمال كل الخصائص والطرق ذات الرؤية عامة أو محمية أو افتراضية والموجودة في الصنف الأهل. لذلك تقدم علاقة التعميم وسيلة عظيمة للتعبير عن أن صنفاً محدداً هو من نوع صنف آخر، وتقدم أيضاً وسيلة لإعادة استعمال الخصائص والسلوكيات بين الأصناف.

علينا الانتباه والتفكير أكثر إذا أردنا استعمال التعميم لمجرد التمكن من إعادة استعمال بعض سلوكيات صنف معيّن. وبما أن الصنف الابن يستطيع رؤية معظم ما بداخل الصنف الأهل فيصبح حينها مقترناً بإحكام بشفرة برمجة الصنف الأهل.

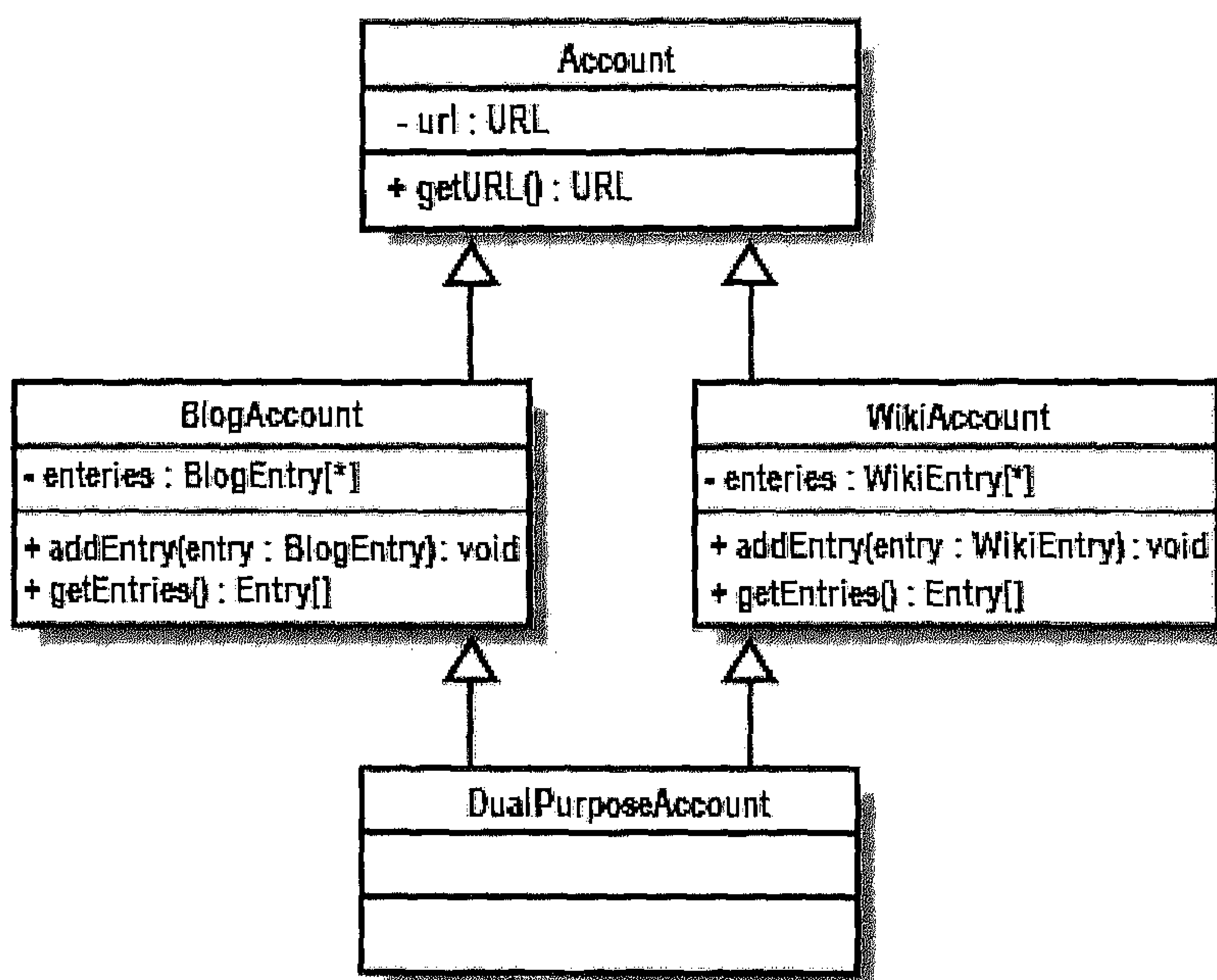
من مبادئ التصميم الكائني التوجه الجيد وتجنب الاقتران المحكم للأصناف، حيث عند التغيير في صنف ما لا نضطر إلى التغيير أيضاً في الأصناف الأخرى. يشكل التعميم الشكل الأقوى للعلاقات بين الأصناف لأنه ينتج اقتراناً محكماً بينها. من هنا، يوجد طريقة مجرّبة جيدة لاستعمال التعميم، وذلك عندما يكون صنف ما هو -فعلياً- نوع أكثر تخصيصاً من صنف آخر، وليس مجرد وسيلة مناسبة لدعم إعادة الاستعمال.



إذا ما زلت تريد إعادة استعمال سلوك صنف ما في صنف آخر، فكر باستعمال التفويض delegation. للمزيد من المعلومات عن كيفية عمل التفويض وسبب تفضيلها على الوراثة، راجع الكتاب المميز، Design Patterns: Elements of Reusable Object-Oriented Software (Addison Wesley).

٥-١-٥-٢. الوراثة المتعدّدة Multiple inheritance

تحصل الوراثة المتعدّدة - أو التعميم المتعدّد في مصطلحات لغة النمذجة الموحدة الرسمية - عندما يرث صنف ما بشكل مباشر من أكثر من صنف أهل، كما هو معروض في الشكل رقم (٥-٩).



شكل رقم (٥-٩) الصنف حساب ثنائي الهدف DualPurposeAccount هو من النوع BlogAccount والنوع WikiAccount مجموعَين معاً لتشكيل نوع واحد.

بالرغم من أن لغة النمذجة الموحدة تدعم الوراثة المتعدّدة، تبقى هذه التقنية غير معتبرة كأفضل ممارسة في معظم الحالات. وهذا بالأساس بسبب المشكلة المعقدة التي تبرز مع الوراثة المتعددة حين تكون الأصناف الأهل لديها تداخل بخصائصها أو سلوكياتها.

وفي الشكل رقم (٥-٩)، يرث الصنف DualPurposeAccount كل سلوكيات و خصائص الصنفَين BlogAccount و WikiAccount، لكن يوجد قليل من التكرار duplication بين الصنفين الأهل. على سبيل المثال، كل من BlogAccount و WikiAccount يحتوي على نسخة من الخاصية name التي قام بوراثتها بدوره من الصنف حساب Account. أيّ نسخة من هذه الخاصية سيأخذ الصنف DualPurposeAccount، أو هل سيأخذ نسختَين من نفس الخاصية؟ تصبح الحالة معقدة أكثر عند احتواء الصنفين الأهل على نفس العملية. كل من الصنف BlogAccount والصنف WikiAccount لديه العملية getEntries().

وبالرغم من فصل الصنف BlogAccount عن الصنف WikiAccount، فلا توجد مشكلة في تواجد العملية getEntries() في كل منهما. على أية حال، ينشأ عندنا تضارب أو تعارض عندما يصبح هذان الصنفان كلاهما أهل لأصناف أخرى من خلال الوراثة المتعددة. عندما يرث DualPurposeAccount من هذين الصنفين معاً، فأي نسخة من الطريقة getEntries() سيأخذ أو سيرث؟ إذا تم استدعاء العملية getEntries() الخاصة بالصنف DualPurposeAccount، فأي طريقة يجب أن تنفذ التي تعطي تدوينات الويكي أو التي تعطي تدوينات المدونة؟

عادة ما تبقى الإجابة عن هذا السؤال - لسوء الحظ - مخفية في التفاصيل البرمجية. على سبيل المثال، إذا كنت تستعمل لغة البرمجة C++

التي تدعم الوراثة المتعددة، ستقوم باستعمال مجموعة قواعد خاصة بلغة C++ حول كيفية حلّ هذه التضاربات. وقد تستعمل لغة برمجة أخرى مجموعة من القواعد المختلفة بالكامل. بسبب هذه التعقيدات، أصبحت الوراثة المتعددة شيئاً كالمحرّم في التطوير الكائنية التوجه للبرامج - إلى درجة أن لغات البرمجة الرائجة حالياً لا تقوم بدعمها، مثل لغة جافا ولغة C#. على أية حال، توجد حالات واقعية حيث يكون معنى للوراثة المتعددة ويمكن برمجتها - في لغات كلغة C++ على سبيل المثال - لذلك تبقى لغة النمذجة الموحدة محتاجة إلى دعمها.

٥-٢ القيود Constraints

قد تريد أحياناً تقييّد أساليب عمل الأصناف. على سبيل المثال، قد تريد تحديد أمرٍ ما يخص الصنف، يسمى ثابت صنف **class invariant**، هذا الثابت عبارة عن قاعدة ما تحدّد أنه يجب على شرط خاص عدم الحدوث أبداً داخل الصنف، أو ارتكاز قيمة خاصية محددة على خاصية أخرى، أو أنه يجب على عملية ما عدم ترك الصنف في حالة غير طبيعية. تتجاوز هذه الأنواع من القيود ما تستطيع عمله مع ترميز بسيط للغة النمذجة الموحدة، و تحتاج إلى لغة بنفسها لهذا الغرض، مثل لغة قيود الكائن Object Constraint Language (OCL).

وهناك ثلاثة أنواع من القيود التي يمكن تطبيقها على أعضاء الصنف باستعمال لغة قيود الكائن:

٥-٢-١ الثوابت Invariants

الثابت هو قيد يجب أن يكون صحيحاً دائماً؛ و إلا فيكون النظام في حالة غير سليمة. ويتم تعريف الثوابت بالارتكاز على خصائص الصنف.

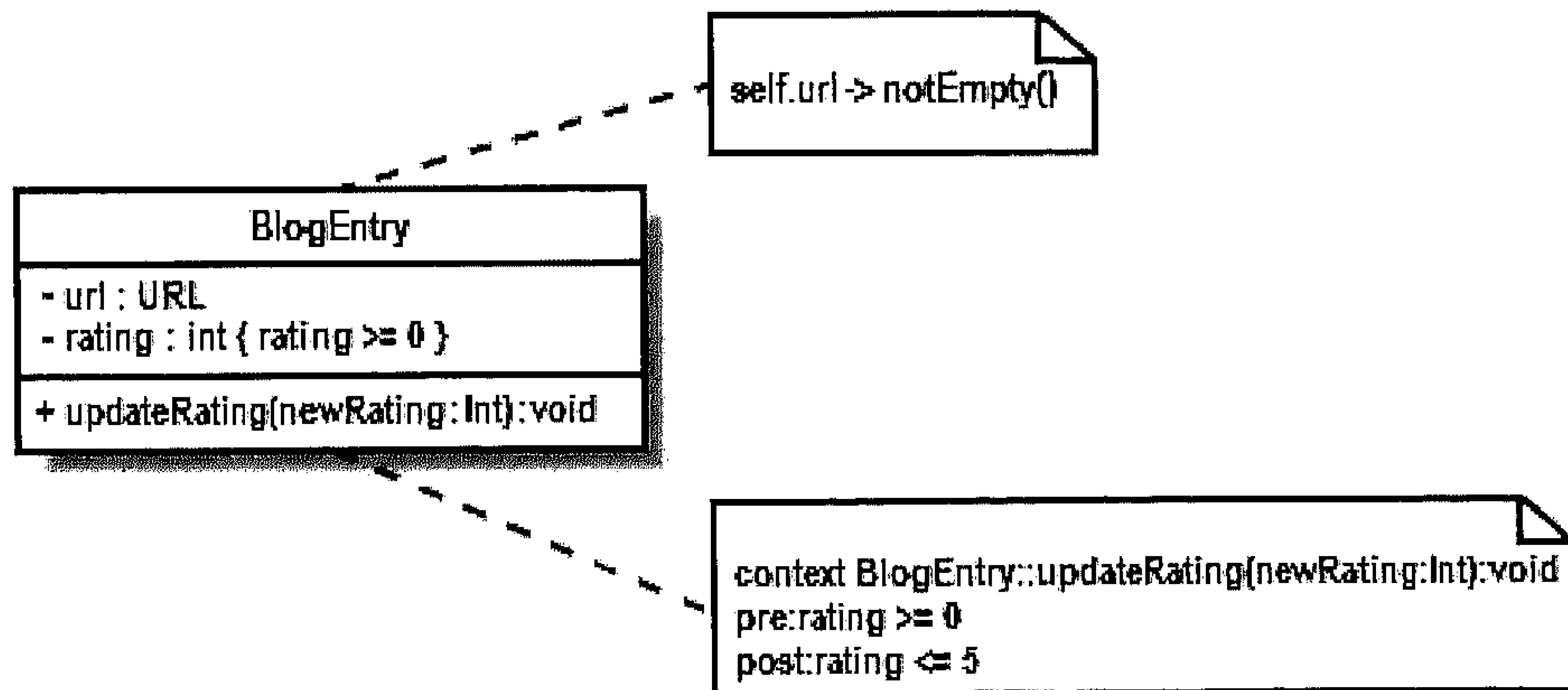
٥-٢-٢ الشروط المسبقة Preconditions

الشرط المسبق هو قيد يتم تعريفه بخصوص طريقة ما، ويتم التأكد من صحته قبل تنفيذ هذه الطريقة. وعادة ما تستعمل الشروط المسبقة للتحقق من صلاحية بارامترات الإدخال الخاصة بالطريقة.

٥-٢-٣ الشروط اللاحقة Postconditions

يتم تعريف الشرط اللاحق أيضاً بخصوص طريقة ما ويتم التأكد من صحته بعد تنفيذ الطريقة. وعادة ما تستعمل الشروط اللاحقة لوصف كيفية تغيير قيم ما من قبل الطرق.

يتم تحديد القيود باستعمال إما تعليمات لغة قيود الكائن OCL داخل القوسين {} بجانب أعضاء الصنف أو في ملاحظة منفصلة، كما هو معروض في الشكل رقم (٥-١٠).



شكل رقم (٥-١٠) تم تحديد ثلاثة قيود على الصنف BlogEntry: القيد self.url > notEmpty() والقيد rating >= 0 من القيود الثابتة، وقيد الشرط اللاحق على العملية updateRating().

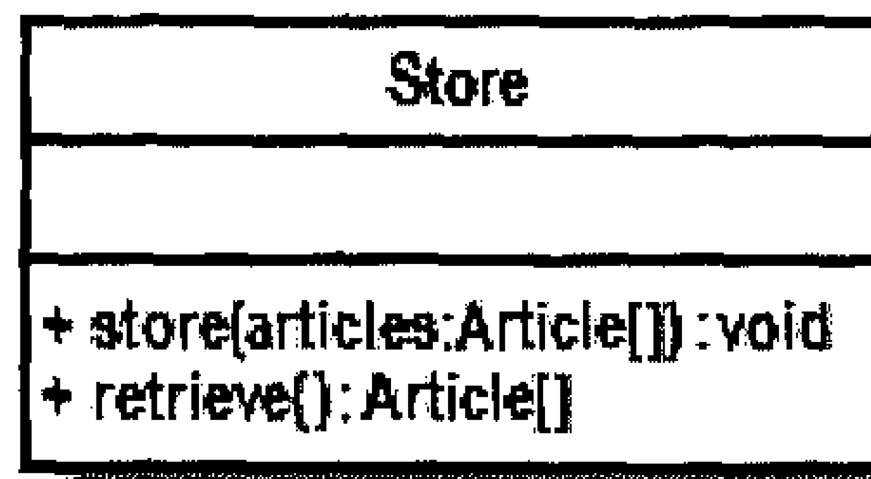
في الشكل رقم (٥-١٠)، تم إجبار الخاصية url كي لا تكون قيمتها أبداً null، وتم إجبار الخاصية (تقدير) rating كي لا تكون أبداً أقل من صفر. ولقد تم وضع شرط مسبق لضمان أن العملية updateRating(..) تقوم باختبار أن الخاصية rating ليست أقل من صفر. أخيراً، يجب على الخاصية rating ألا تصبح أبداً أكبر من خمسة بعد أن يتم تحديثها، بالتالي تم تحديد ذلك كقيد شرط لاحق على العملية updateRating (..).



تسمح لغة قيود الكائن OCL بتحديد كل أنواع القيود التي تقيّد كيفية عمل الأصناف. انظر إلى الملحق أ للمزيد من المعلومات حول لغة قيد الكائن.

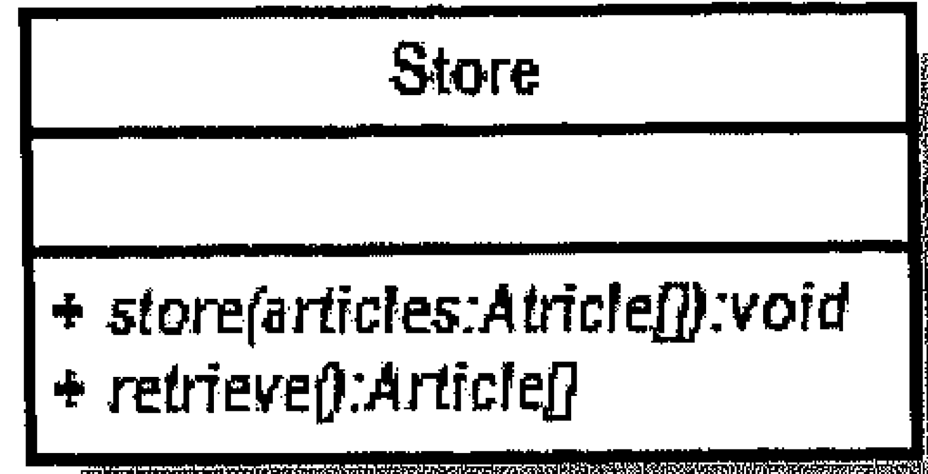
٥-٣ الأصناف المجردة Abstract classes

عند استعمال التعميم للتصريح عن صنف عام قابل لإعادة الاستعمال، لن تقدر أحيانا على برمجة كل السلوكيات التي يحتاجها هذا الصنف. وإذا كنت تبرمج الصنف مخزن **Store** لتخزين واسترجاع مقالات نظام إدارة المحتوى، كما هو معروض في الشكل (٥-١١)، فقد تريد الإشارة عند هذه النقطة إلى أن المخزن **Store** لا يعرف كيفية تخزين واسترجاع المقالات بالضبط و يجب إهمال هذا الأمر كي تقرر الأصناف الفرعية كيفية إنجاز ذلك.



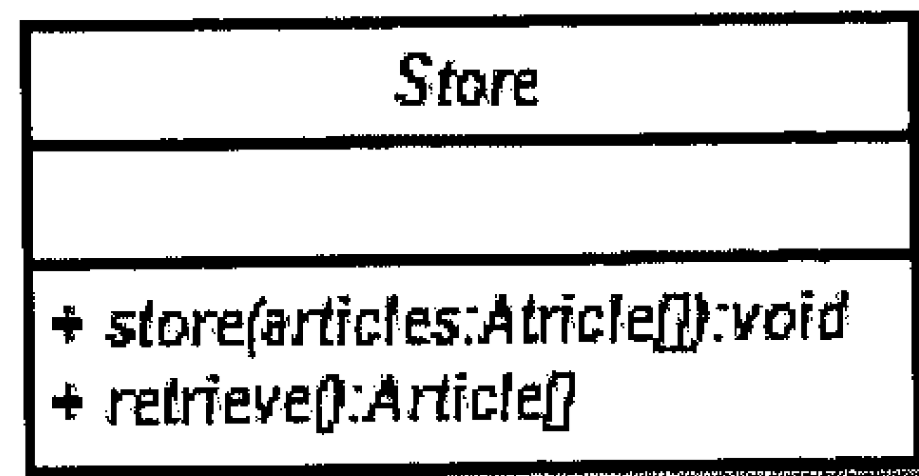
شكل رقم (٥-١١) باستخدام العمليات العادية، يحتاج الصنف مخزن **Store** إلى معرفة كيفية تخزين و استرجاع مجموعة المقالات.

للإشارة إلى أن برمجة العمليتين (..)store و (..)retrieve ستترك إلى الأصناف الفرعية، نقوم بالتصريح عنها كمجردة حيث يتم ذلك بكتابة توقيعها signatures بالطراز الإيطالي italic style، كما هو معروض في الشكل رقم (٥-١٢).



شكل رقم (٥-١٢) لا تحتاج العمليتان (..)store و (..)retrieve الآن إلى أن تكون مبرمجة من قبل الصنف Store.

لا تحتوي العملية المجردة على شفرة برمجة، وهي حقيقة مكان للإعلان التالي "سأترك برمجة هذا السلوك إلى أصنافي الفرعية". إذا تم التصريح عن أي جزء من الصنف على أنه مجرد، فمن الضروري أيضاً التصريح عن الصنف نفسه على أنه مجرد، ويتم ذلك بكتابة اسمه بالطراز الإيطالي، كما هو معروض في الشكل رقم (٥-١٣).



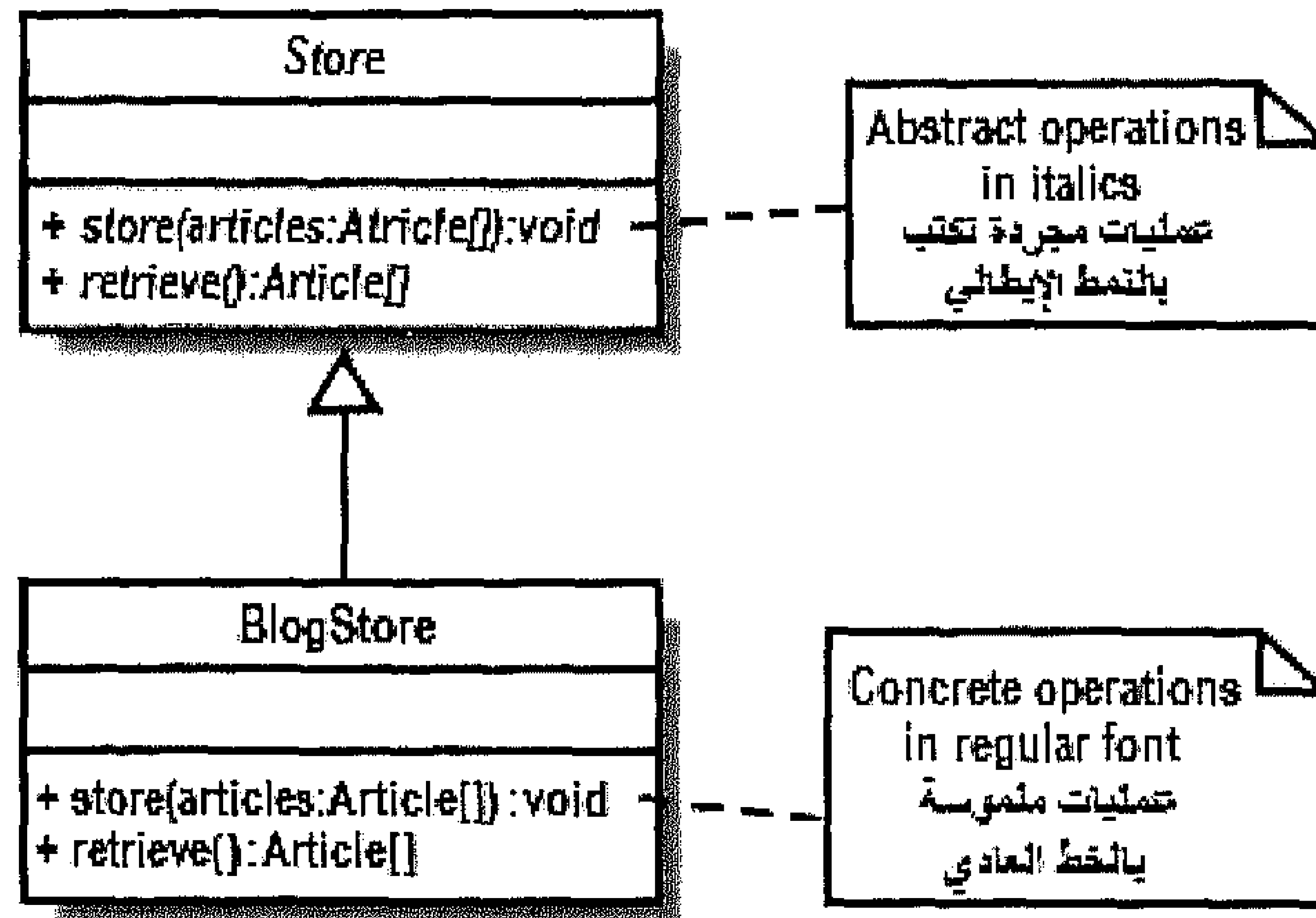
شكل رقم (٥-١٣) الصنف المجرد بالكامل مخزن Store.

بعد التصريح الآن عن العمليتين (..)store و (..)retrieve في الصنف Store على أنهما مجردتان، فليس عليهما احتواء أي شفرة برمجة، كما هو معروض في المثال رقم (٥-٤).

مثال رقم (٥-٤) لقد تم حل مشكلة تحديد أي شفرة برمجة يجب و ضعها في إنجاز العمليتين store() و retrieve() بالتصريح عنهما و عن الصنف المحيط بهما مع خاصية التجريد abstract.

```
public abstract class Store {
   public abstract void store(Article[] articles);
   public abstract Article[] retrieve();
}
```

لا يمكن إنشاء كائنات مثيلة لصنف مجرد، لأن لديه أجزاء ناقصة لم يتم تعريفها بالكامل. وربما يقوم الصنف Store ببرمجة شفرة العمليتين store(..) و retrieve(..) لكن بما أنه صنف مجرد، يجب على الأصناف الأبناء التي ترث منه أن تبرمج شفرة عملياته المجردة أو أن تصرح عنها بأنها مجردة، كما هو معروض في الشكل رقم (٥-١٤).



شكل رقم (٥-١٤) يرث الصنف مخزن مدونة BlogStore من الصنف المجرد مخزن Store و يقوم ببرمجة العمليتين store(..) و retrieve(..)؛ وتتم الإشارة إلى الأصناف التي تبرمج كلياً كل العمليات المجردة الموروثة من أهلها بأنها أصناف ملموسة concrete.

وعندما أصبح الصنف Store مجرداً، تم تأخير برمجة العمليتين store(..) و retrieve(..) حتى يصبح لدى صنف فرعي معلومات كافية لبرمجتهما. ويستطيع الصنف BlogStore برمجة عمليات الصنف Store المجردة؛ لأنه يعرف كيف يخزن المدونة، كما هو معروض في المثال رقم (٥-٥).

مثال رقم (٥-٥) يقوم الصنف **BlogStore** بتكملة الأجزاء المجردة في الصنف **Store**.

```
public abstract class Store {
   public abstract void store(Article[] articles);
   public abstract Article[] retrieve();
}
public class BlogStore {
   public void store(Article[] articles) {
      // خزّن جانباً تدوينات المدونة هنا...
   }
   public Article[] retrieve() {
      // استخرج وارجع تدوينات المدونة المخزنة هنا...
   }
}
```

لا يمكن إنشاء كائن مثيل لصنف مجرد، لأن هناك أجزاء ناقصة من تعريف الصنف: الأجزاء المجردة. يمكن إنشاء كائنات مثيلة للأصناف الأبناء لصنف مجرد شرط أن تقوم بتكملة تعريف كل الأجزاء المجردة التي في الصنف الأهل، وتكون بالتالي قد أصبحت أصنافاً ملموسة، كما هو معروض في المثال رقم (٥-٦).

تشكل الأصناف المجردة آلية قوية جداً لتعريف السلوكيات والخصائص المشتركة، لكنها تترك بعض جوانب كيفية عمل الصنف إلى الأصناف الفرعية الملموسة أكثر. وكمثال شهير عن مكان استعمال

الأصناف المجردة والواجهات هو عند تعريف الوظائف و السلوكيات العامة المؤلفة تصميم الأنماط design patterns. وعلى أية حال، يجب استعمال الوراثة لإنجاز الأصناف المجردة؛ لذلك تحتاج إلى الإلمام بكل نواحي علاقة التعميم القوية والمحكمة الاقتران.

مثال رقم (٥-٦) يمكن إنشاء كائنات من الأصناف غير المجردة **non-abstract**، و يحتاج أي صنف مصرح عنه غير مجرد إلى برمجة أي سلوك مجرد موروث.

```
public abstract class Store { // الصنف مخزن
   public abstract void store(Article[] articles);
   public abstract Article[] retrieve();
}
public class BlogStore { // الصنف مخزن مدونة
   public void store(Article[] articles) {
      // خزّن جانباً تدوينات المدونة هنا...
   }
   public Article[] retrieve() {
      // استخرج وارجع تدوينات المدونة المخزنة هنا...
   }
}
public class MainApplication {
   public static void main(String[] args) {
      // إنشاء كائن مثيل للصنف مخزن مدونة
      // هذا جيد كلياً لأن الصنف مخزن مدونة غير مجرد
      BlogStore store = new BlogStore();
      blogStore.store(new Article[]{new BlogEntry()});
      Article[] articlesInBlog = blogStore.retrieve();
      // مشكلة: لا يوجد معنى لإنشاء كائن من صنف مجرد
      // لأنه لم تتم برمجة شفرات الأجزاء المجردة بعد
     Store store = new Store(); // ينتج خطأ ترجمة هنا
   }
}
```

انظر إلى القسم السابق في هذا الفصل "التعميم (المعروف على نحو آخر بالوراثة)" للمزيد من المعلومات عن تجارب و مِحَن استعمال التعميم.

للمزيد عن تصميم الأنماط وكيفية إحسان استعمال الأصناف المجردة، راجع الكتاب Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley).

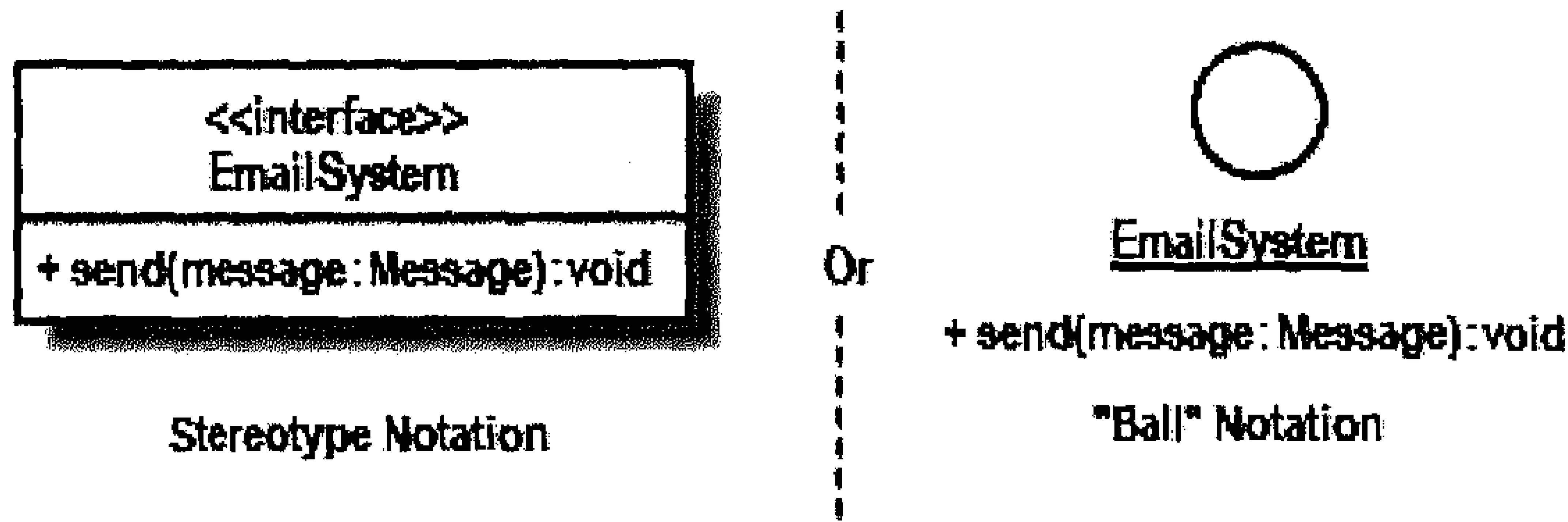
٥-٤ الواجهات Interfaces

إذا أردت التصريح عن الطرق التي يجب على الأصناف الملموسة برمجتها، لكن من دون استعمال التجريد بسبب عدم توفر الوراثة المباشرة المتعددة (مثل لغة جافا)، يمكن بالتالي استعمال الواجهات لهذا الغرض.

الواجهة هي مجموعة عمليات غير معرف لها شفرة برمجة، وهي تشبه – كثيراً – الصنف المجرد الذي يحتوي على طرق مجردة فقط. في بعض لغات البرمجة، مثل لغة C++، وتبرمج الواجهات باستعمال أصناف مجردة لا تحتوي على شفرة برمجة للعمليات. في لغات البرمجة الأحدث، مثل جافا وC#، للواجهة تركيبة بنيوية خاصة بها.

فكر بالواجهة كأنها عقد بسيط جداً يعلن، "تلك هي العمليات التي يجب أن تبرمجها الأصناف العازمة على استيفاء هذا العقد". بالإضافة إلى ذلك قد تحتوي الواجهة أحياناً على خصائص، لكن في تلك الحالات، عادة ما تكون تلك الخصائص ساكنة وغالباً ما تكون ثابتة. (انظر إلى الفصل الرابع للمزيد عن استعمال الخصائص الساكنة).

في لغة النمذجة الموحدة، يمكن عرض الوجهة باستعمال ترميز صنف له حاشية، أو باستعمال ترميز الكرة الخاص بها، كما هو معروض في الشكل رقم (٥-١٥).



شكل رقم (٥-١٥) أسر واجهة لنظام رسائل **EmailSystem** باستعمال ترميز الحاشية وترميز "الكرة" في **UML**؛ بخلاف الأصناف المجردة، ليس على الواجهة إظهار عدم إنجاز عملياتها، لذلك ليس هناك حاجة لكتابتها باستعمال الطراز الإيطالي.



تميل الوجهات لتكون أكثر أماناً في الاستعمال من الأصناف المجردة لتفاديها عدداً من المشاكل المرتبطة بالوراثة المتعددة (انظر سابقاً إلى قسم "الوراثة المتعددة" في هذا الفصل). لهذا السبب تسمح لغات البرمجة كلغة جافا للصنف بإنجاز أي عدد من الواجهات، لكن يمكن للصنف الوراثة المباشرة من صنف عادي أو مجرد واحد فقط.

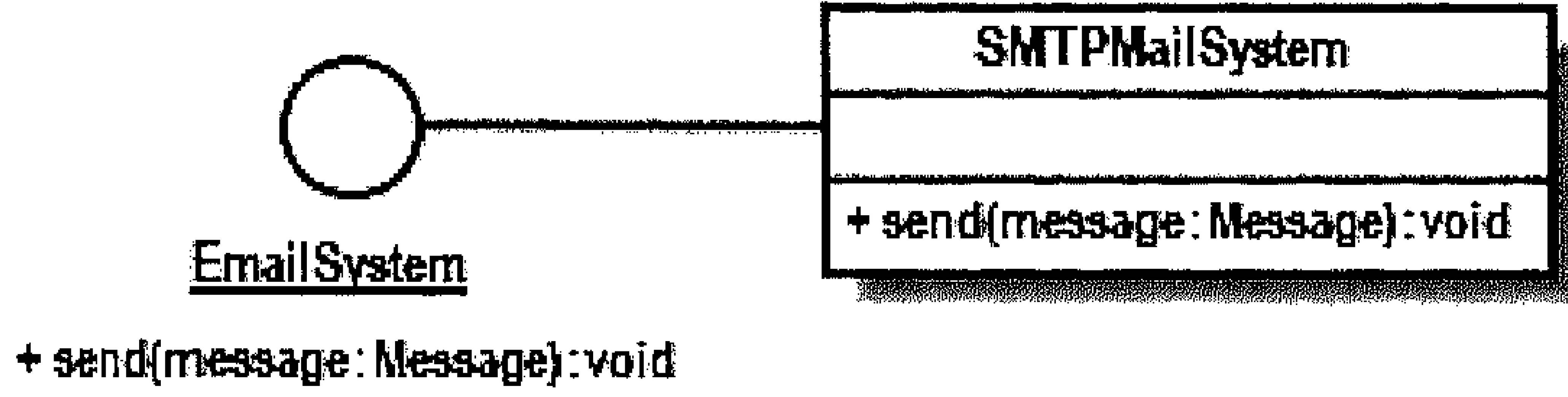
إذا أردت إنجاز الواجهة EmailSystem التي في الشكل رقم (٥-١٥) بلغة جافا، ستبدو شفرتها كما في المثال رقم (٥-٧).

مثال رقم (٥-٧) يتم إنجاز الواجهة EmailSystem بلغة جافا باستعمال الكلمة المفتاح interface، وتحتوي على توقيع العملية send(..) فقط من دون إنجازها.

```
public interface EmailSystem {
    public void send(Message message);
}
```

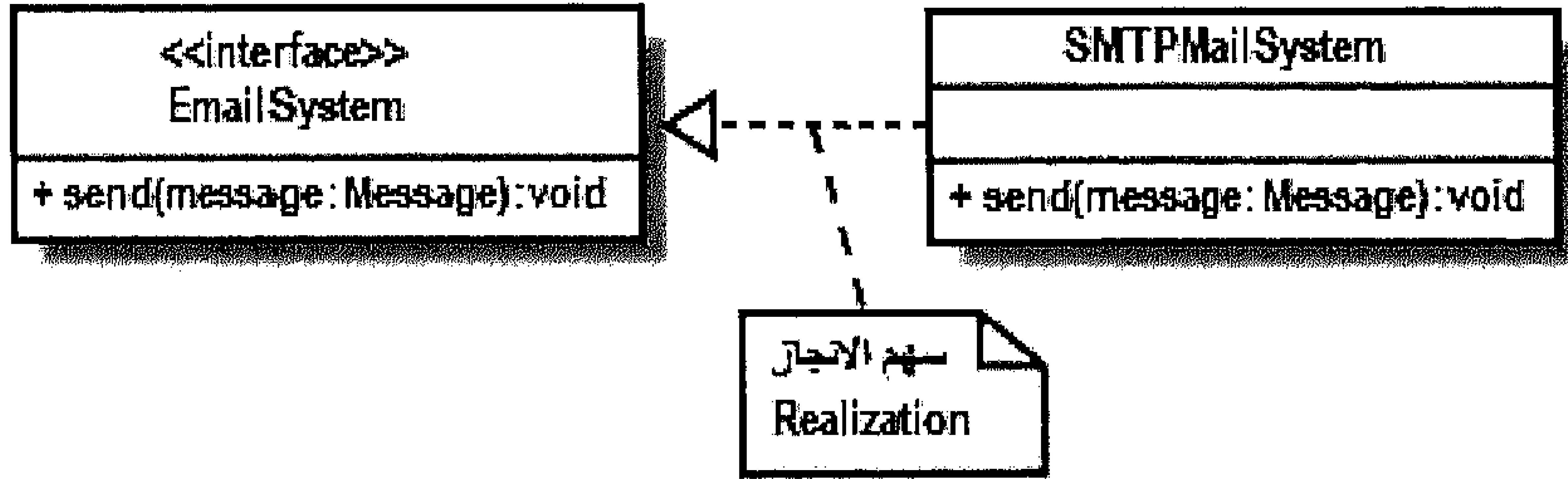
لا نستطيع إنشاء كائن من الواجهة نفسها، كما أننا لا نستطيع إنشاء كائن مثيل لصنف مجرد. هذا بسبب غياب كل إنجازات عمليات الواجهة حتى إنجازها من قبل صنف ما. إذا قمت باستعمال ترميز "الكرة"

للواجهة، يتم بالتالي إنجاز الواجهة من خلال ربطها بصنف ما، كما هو معروض في الشكل رقم (٥-١٦).



شكل رقم (٥-١٦) يُنجز الصنف **SMTPMailSystem** كل العمليات المحددة في الواجهة **EmailSystem.**

إذا قمت باستعمال ترميز الحاشية للواجهة، تحتاج بالتالي إلى سهم جديد لإظهار أنه عندنا علاقة إنجاز realization، كما هو معروض في الشكل رقم (٥-١٧).



شكل رقم (٥-١٧) يحدد سهم الإنجاز بأن الصنف **SMTPMailSystem** ينجز الواجهة **EmailSystem.**

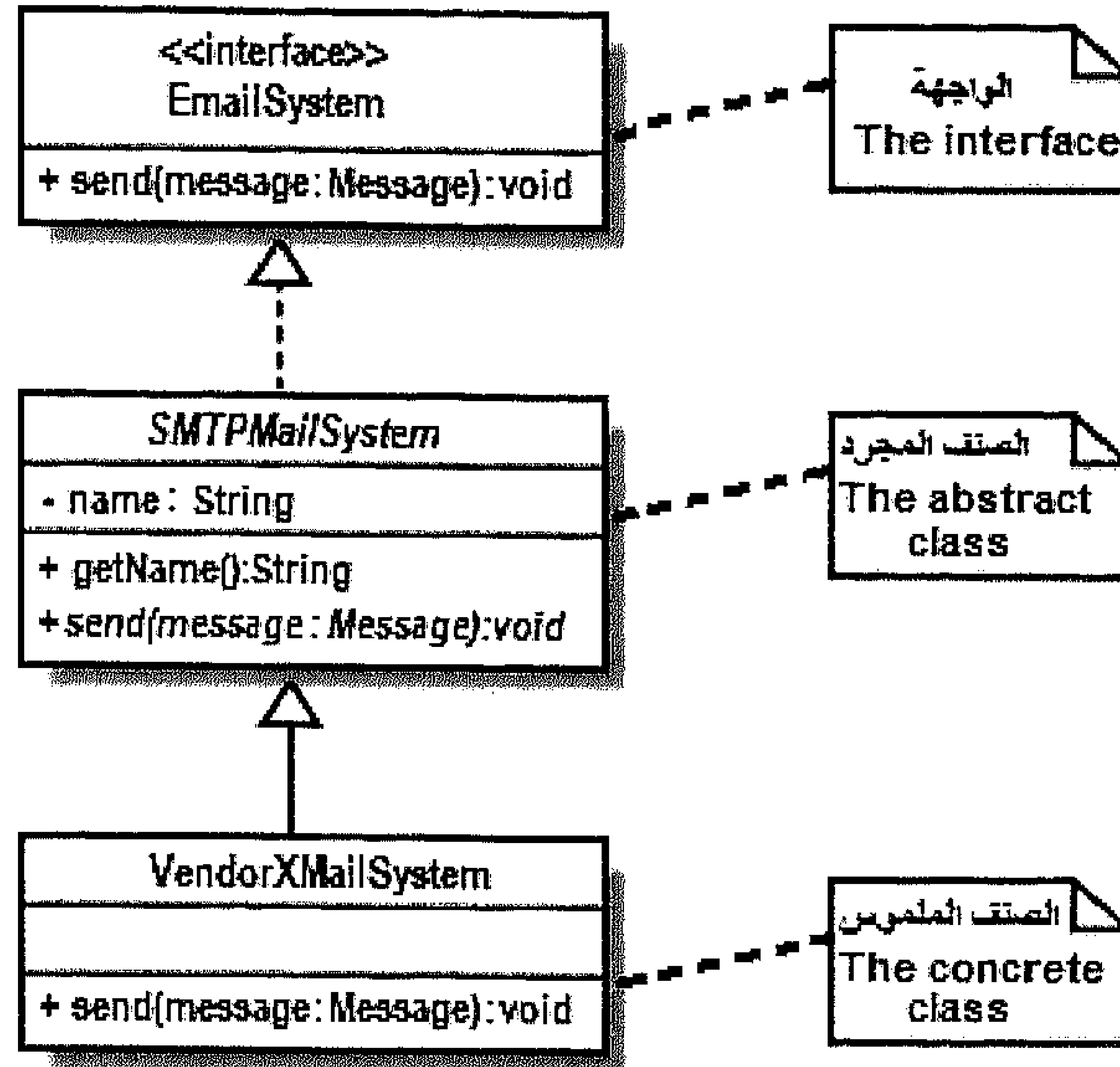
ينتج كلا الشكلَين رقم (٥-١٦) ورقم (٥-١٧) نفس شفرة المصدر بجافا بسبب توليدها بشكل آلي، كما هو معروض في المثال رقم (٥-٨).

مثال رقم (٥-٨) تُنجز أصناف جافا الواجهات باستعمال الكلمة المفتاح **implements**.

```
public interface EmailSystem {
   public void send(Message message));
}
public class SMTPMailSystem implements EmailSystem {
    public void send(Message message) {
       // أنجز التفاعلات مع خادم أس أم تي بي لإرسال الرسالة //
    }
   //... SMTPMailSystem إنجازات العمليات الأخرى التي في الصنف //
}
```

إذا قام صنف ما بإنجاز واجهة محددة لكن من دون برمجة كل الطرق المحددة فيها، فيجب التصريح عن هذا الصنف بأنه مجرد، كما هو معروض في الشكل رقم (٥-١٨).

تبرز عظمة الواجهات من خلال فصلها التام للسلوك المطلوب من الصنف عن كيفية برمجته بالضبط. عندما يقوم صنف ما بإنجاز واجهة محددة، ويمكن الإشارة إلى كائنات هذا الصنف باستعمال اسم الواجهة بدلاً من استعمال اسم الصنف نفسه. هذا يعني إمكانية اعتماد أصناف أخرى على الواجهات بدلاً من اعتمادها على الأصناف. وهذا الأمر جيد في العموم بسبب ضمانه حرية اقتران الأصناف قدر الإمكان. إذا اقترنت الأصناف بحرية، ويجب ألا يتسبب تغيير برمجة صنف ما بمشكلة للأصناف الأخرى (بسبب اعتمادهما على الواجهة وليس على الصنف نفسه).



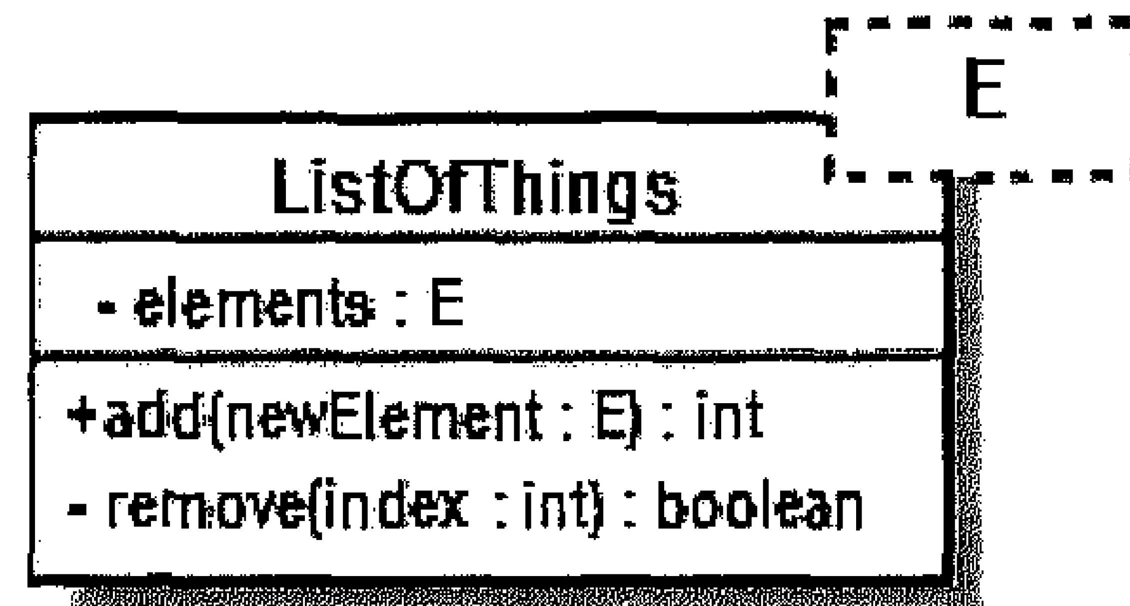
شكل رقم (٥-١٨) يجب التصريح عن الصنف SMTPMailSystem بأنه مجرد بسبب عدم برمجته العملية send(..) المحددة في الواجهة EmailSystem؛ يكتمل الوصف ببرمجة الصنف VendorXMailSystem لكل عمليات الواجهة.

استعمال الواجهات

يعتبر فك اقتران التبعيّات بين الأصناف باستعمال الوجهات ممارسة جيدة؛ تقوّي بعض بيئات البرمجة العلاقة صنف - واجهة، مثل Spring FrameWork. يفيد استعمال الواجهات كمقابل الأصناف المجردة أيضاً في برمجة تصميم الأنماط. في لغات البرمجة مثل جافا، لا تريد حقاً استعمال علاقة الوراثة الفردية لمجرد استعمال نمط التصميم. ويمكن لأي صنف جافا إنجاز أي عدد من الواجهات، لذلك توفر الواجهات وسيلة لتنفيذ نمط التصميم من دون فرض عبء وجوب استهلاك علاقة الوراثة الفردية للقيام بذلك الأمر.

٥-٥ القوالب Templates

القوالب هي ميزة متقدمة لكن مفيدة من الأسلوب الكائني التوجه. وتفيد القوالب عندما تريد تأجيل قرار تحديد الأصناف التي سيعمل عليها صنفاً ما. يشار للقالب أحياناً بالصنف ذات بارامترا. يشبه التصريح عن القالب القول "أعرف أن على هذا الصنف العمل مع أصناف أخرى، لكني لا أعرف أو بالضرورة لا أهتم بما ستكون عليه هذه الأصناف فعلياً، كما هو معروض في الشكل رقم (٥-١٩).

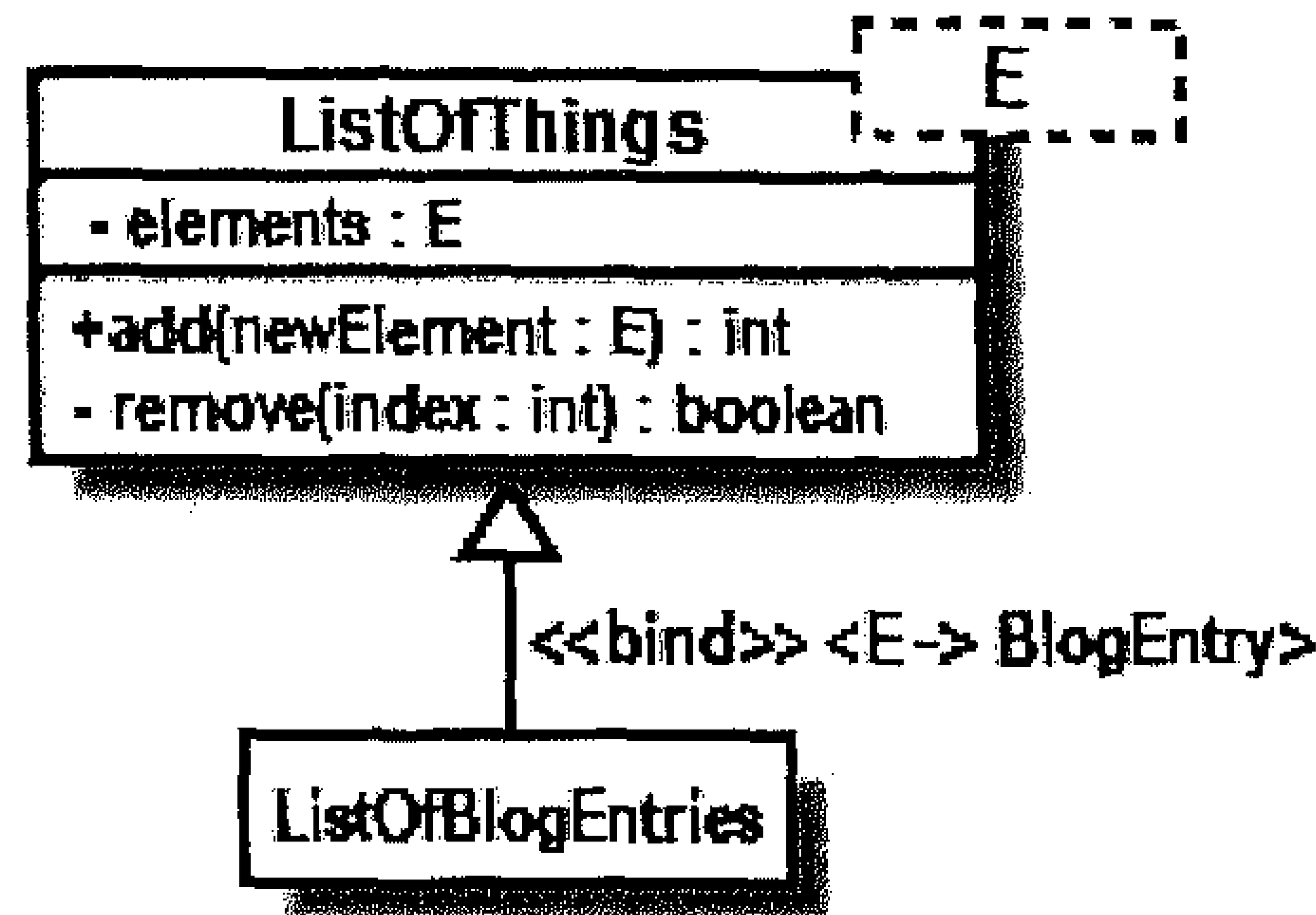


شكل رقم (٥-١٩) يعرض القالب في UML كصندوق الصنف العادي مضاف إليه بأعلى اليمين صندوق منقّط الأطراف.

الصنف ListOfThings في الشكل رقم (٥-١٩) هو صنف ذات بارامتر من النوع E. ولا يوجد صنف يدعى E في النموذج؛ والحرف E ليس أكثر من مكان احتواء يتم استعماله لاحقاً لإخبار الصنف ListOfThings عن نوع الكائن الذي يحتاج إلى تخزينه.

ولاستعمال الصنف الذي يكون قالباً، تحتاج أولاً إلى ربط البارامترات الخاصة به. ولا يزال الصنف القالب ListOfSomething يجهل ما المفروض عليه تخزينه؛ ونحتاج إلى تحديد الأصناف الفعلية التي سيعمل عليها القالب؛ نحتاج فقط إلى ربط بارامتر القالب (المشار إليه بالحرف E) بصنف فعلي.

يمكن ربط بارامترات القالب بمجموعة محددة من الأصناف من خلال إحدى الوسيلتين. أولاً، يمكن إنشاء صنف فرعي للقالب وربط البارامترات خلال هذا الإنشاء، كما هو معروض في الشكل رقم (٥-٢٠).



شكل رقم (٥-٢٠) تم إنشاء الصنف الفرعي **ListOfBlogEntries** للصنف **ListOfThings**، مع ربط البارامتر الوحيد **E** بالصنف الملموس **BlogEntry**.

يسمح الربط بصنف فرعي، كما في الشكل رقم (٥-٢٠)، بإعادة استعمال كل السلوكيات العمومية التي في الصنف ListOfThings، وحصر السلوكيات التي في الصنف ListOfBlogEntries للقيام فقط بإضافة و إزالة كائنات BlogEntry.

تتجلى القوة الحقيقية للقوالب أكثر بكثير عند استعمال الوسيلة الثانية لربط بارامترات القوالب التي تتم عند وقت التشغيل. ويتم ربط بارمتر القالب عند وقت التشغيل عند عملية إنشاء كائن من القالب وإخباره بالأنواع الفعلية لبارامتراته.

ويخص ربط القالب عند وقت التشغيل الكائنات و ليس الأصناف؛ لذلك يتطلب وجود نوع جديد من المخططات؛ ألا وهو مخطط الكائنات. وتقوم مخططات الكائنات باستعمال الأصناف لإظهار بعض الوسائل

المهمة المستعملة خلال تنفيذ النظام. وستكون مخططات الكائنات موضوع الفصل القادم.

القوائم Lists

تميل القوائم لتكون الأمثلة الأكثر انتشاراً عن كيفية استعمال القوالب. وتقوم القوائم والأنواع القريبة منها (مثل الخرائط maps والمجموعات sets) بتخزين كائنات بأساليب متنوعة، لكنها لا تهتم فعلياً بأصناف تلك الكائنات. لهذا السبب، وأحد أفضل الاستعمالات الواقعية للقوالب هو مع أصناف المجموعات في جافا java collection. قبل الإصدار ٥ لجافا، لم يكن لديها وسائل لتحديد القوالب. مع إطلاق الإصدار ٥ لجافا الذي يضم ميزة التعميم generics، أصبح بإمكانك إنشاء القوالب الخاصة بك، وكما أصبحت كل أصناف المجموعات الأساسية في جافا متوفرة أيضاً للاستعمال كقوالب. للتعلم أكثر حول التعميم في جافا ٥، راجع أخر نسخة من الكتاب Java in a Nutshell (O'Reilly).

٥-٦ ما هي الخطوة التالية؟

تعرض مخططات الأصناف أنواع الكائنات التي في النظام. كخطوة تالية مفيدة، ويمكن النظر إلى مخططات الكائنات؛ لأنها تعرض كيف تصبح الأصناف حيّة عند وقت التشغيل كمثيلات كائنية، و التي تكون مفيدة في إظهار ترتيبات configurations وقت التشغيل. وستتم تغطية مخططات الكائنات في الفصل السادس.

إن الهياكل المركبة هي نوع مخططات تظهر بحرية مخططات الأصناف الحساسة السياق والأنماط التي في البرامج. وستتم تغطية الهياكل المركبة في الفصل الحادي عشر.

بعد القيام بتحديد مسؤوليات الأصناف في النظام، من الشائع إنشاء مخططات التتابع والاتصال لإظهار التفاعلات التي بين الأجزاء. ويمكن أن تجد مخططات التتابع في الفصل السابع؛ وستتم تغطية مخططات الاتصال في الفصل الثامن.

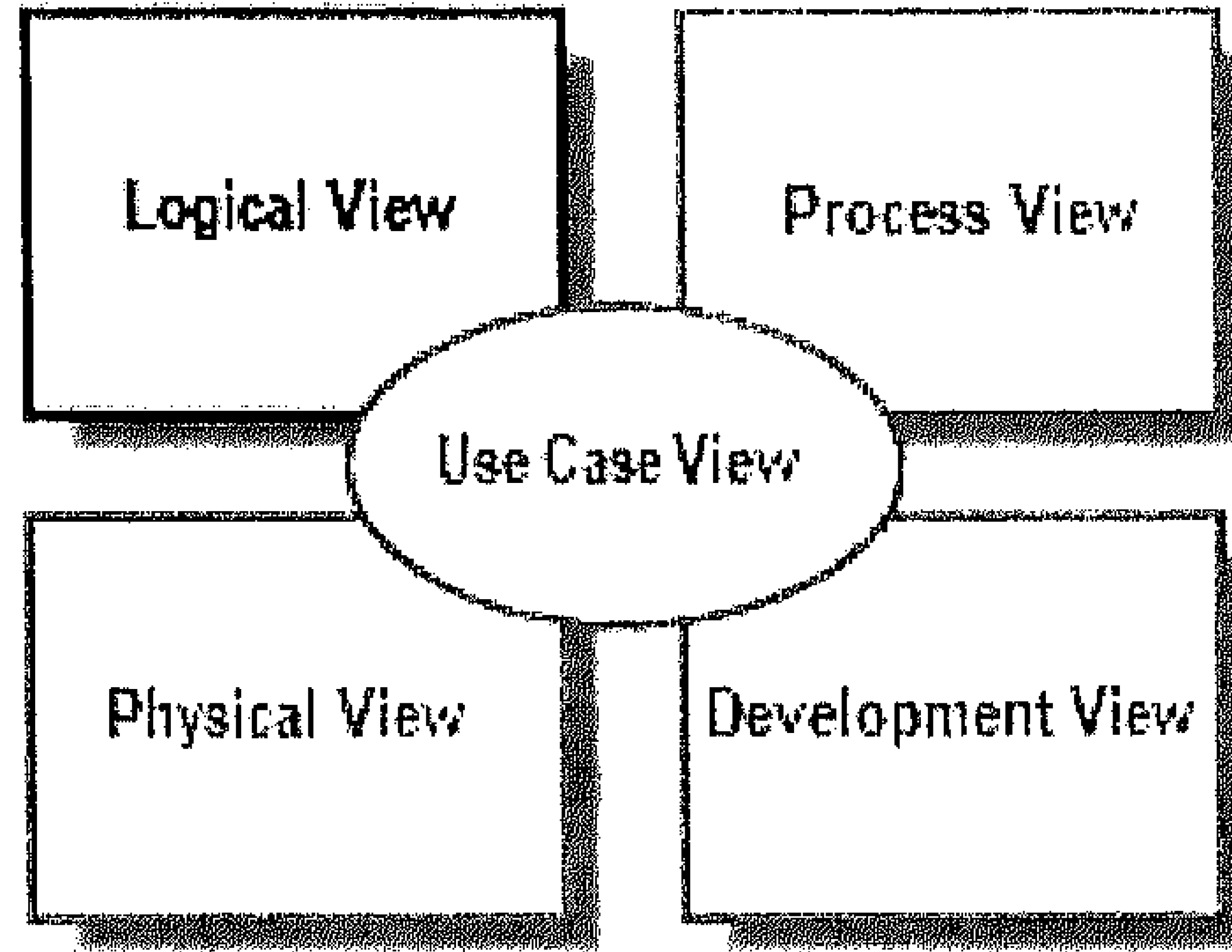
من الشائع أيضاً الرجوع لتنظيم الأصناف في حُزَم. وتسمح لك مخططات الحزم برؤية التبعيات بمستوى أعلى، ويساعدك هذا على فهم استقرار البرامج. وستتم تغطية مخططات الحزم في الفصل الثالث عشر.

نقل الأصناف إلى الحياة:
مخططات الكائنات

BRINGING YOUR CLASSES TO LIFE: OBJECT DIAGRAMS

تشكل الكائنات قلب أي نظام كائني التوجه عند وقت التشغيل. وعند الاستعمال الفعلي للنظام الذي تم تصميمه، تكون أجزاؤه مؤلفة من الكائنات التي تنقل كل الأصناف المصممة بعناية إلى الحياة.

ويعتبر ترميز مخطط الكائنات بسيط جداً بالمقارنة مع مخططات الأصناف. ورغم امتلاكه عدداً محدوداً من المفردات الواضحة، فإنَّ مخططات الكائنات تفيد بشكل خاصة في وصف كيفية عمل الكائنات سوياً داخل النظام ضمن سيناريو معيّن. وتصف مخططات الأصناف كيفية تفاعل الأنواع المختلفة للكائنات التي داخل النظام مع بعضها بعضا. وتجذب الانتباه أيضاً إلى الأشكال العديدة للكائنات وتفاعلها داخل النظام عند وقت التشغيل. بالإضافة إلى مخططات الأصناف، تفيد مخططات الكائنات في أسر المنظور المنطقي للنموذج والمعروض في الشكل رقم (٦-١).



شكل رقم (٦-١) يحتوي المنظور المنطقي للنموذج على التوصيفات المجردة لأجزاء النظام التي تتضمن الكائنات الموجودة داخل النظام عند وقت التشغيل.

٦-١ مثيلات الكائن Object Instances

كي نتمكن من رسم مخطط الكائنات، نحتاج أولاً إلى إضافة الكائنات نفسها. ويرمز إلى الكائنات بأسلوب بسيط جداً يشبه كثيراً ترميز الأصناف؛ ويتم عرض الكائن باستعمال شكل المستطيل كما مع الصنف، لكن يتم التسطير تحت اسم الكائن لتبيان أنه مثيل للصنف وليس الصنف نفسه، كما هو معروض في الشكل رقم (٦-٢).



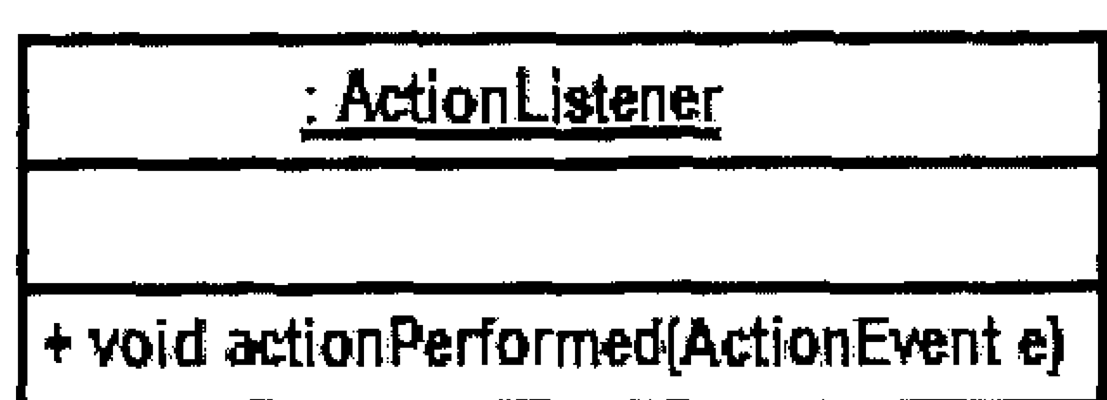
شكل رقم (٦-٢) إظهار كائن مثيل باستعمال المستطيل و التسطير تحت اسمه.

يتميز الكائن entry الذي في الشكل رقم (٦-٢) بهويته فقط (الاسم entry) التي تستعمل للإشارة إلى الكائن. على أية حال، إذا كنت تعرف الصنف الذي يكون الكائن مثيلاً له، يمكنك أيضاً تحديد اسم الصنف مع اسم الكائن، كما هو معروض في الشكل رقم (٦-٣).

entry : BlogEntry

شكل رقم (٦-٣) الكائن entry هو مثيل للصنف BlogEntry من الفصل الرابع.

يوجد نوع أخير من الكائنات التي يمكن استعمالها في مخطط الكائنات ألا و هو الكائن مجهول الاسم anonymous، انظر إلى الشكل رقم (٦-٤).



شكل رقم (٦-٤) صنف الكائن المجهول هو ActionListener، لكن لم يتم تحديد اسم أو هوية؛ يقدم أيضاً هذا الكائن الطريقة الوحيدة actionPerformed(..) المطلوبة من قبل الواجهة ActionListener.

عادة ما تفيد الكائنات مجهولة الاسم عندما يكون الاسم غير مهم ضمن السياق المستعمل فيه. وعلى سبيل المثال، هناك اصطلاح برمجي شائع لاستعمال كائن مجهول الاسم، وذلك عند إنشاء كائن مدير حدث event handler في جافا، ولا نهتم هنا باسم الكائن بل نهتم فقط بتسجيل الكائن مع مصدر الحدث الملائم، كما هو معروض في المثال رقم (٦-١).



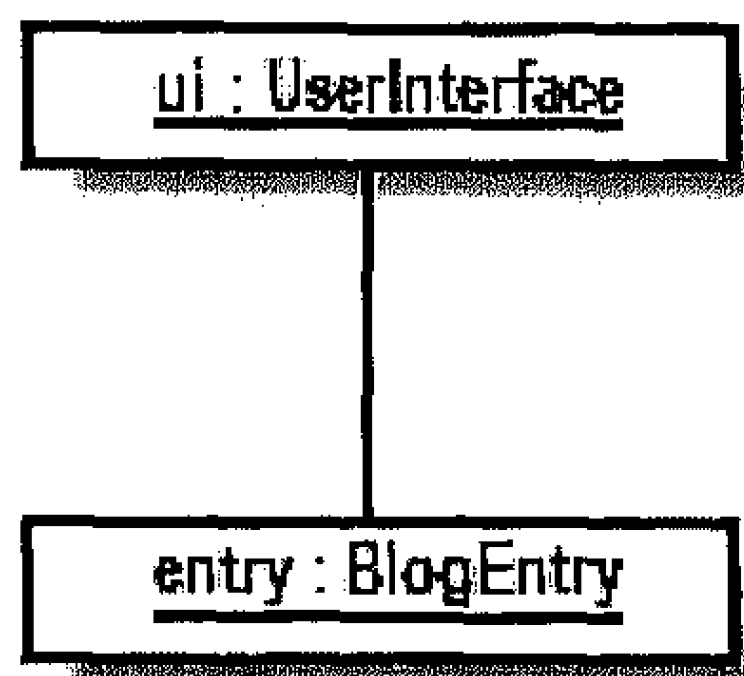
في المثال رقم (٦- ١)، يمكن أيضاً ملاحظة قيام الكائن مجهول الاسم بإنجاز الواجهة ActionListener كما هو مصرّح عنه. بينما يتم إنشاء الكائن مجهول الاسم، وتتم برمجة شفرة الطريقة actionPerformed(..) المطلوبة من قبل الواجهة ActionListener. وتم استعمال صيغة لغة جافا في كتابة الطريقة.

مثال رقم (٨-١) استعمال كائن مجهول الاسم في برنامج جافا لتسجيل مستمع إلى حدث من النوع **ActionListener** (يصدر الحدث عن زر من الصنف **JButton**).

```
public void initialiseUI( ) {
  //... برمجة شفرة الطرق الأخرى...
  JButton button = new Jbutton("Submit");
  button.addActionListener(
    new ActionListener {
      public void actionPerformed(ActionEvent e) {
        System.out.print("The button was pressed so it's time");
        System.out.println("to do something...");
        //.. تم النقر على الزر وحان وقت عمل شيء ما....
      }
    } );
 //... برمجة شفرة الطرق الأخرى...
}
```

٦-٢ الروابط Links

لا تكون الكائنات مهمة أو مفيدة بحد ذاتها. نحتاج إلى الروابط لربط الكائنات سوية ولإظهار كيفية استعمالها معاً في ترتيبات معينة configuration عند وقت التشغيل، كما هو معروض في الشكل رقم (٦-٥).

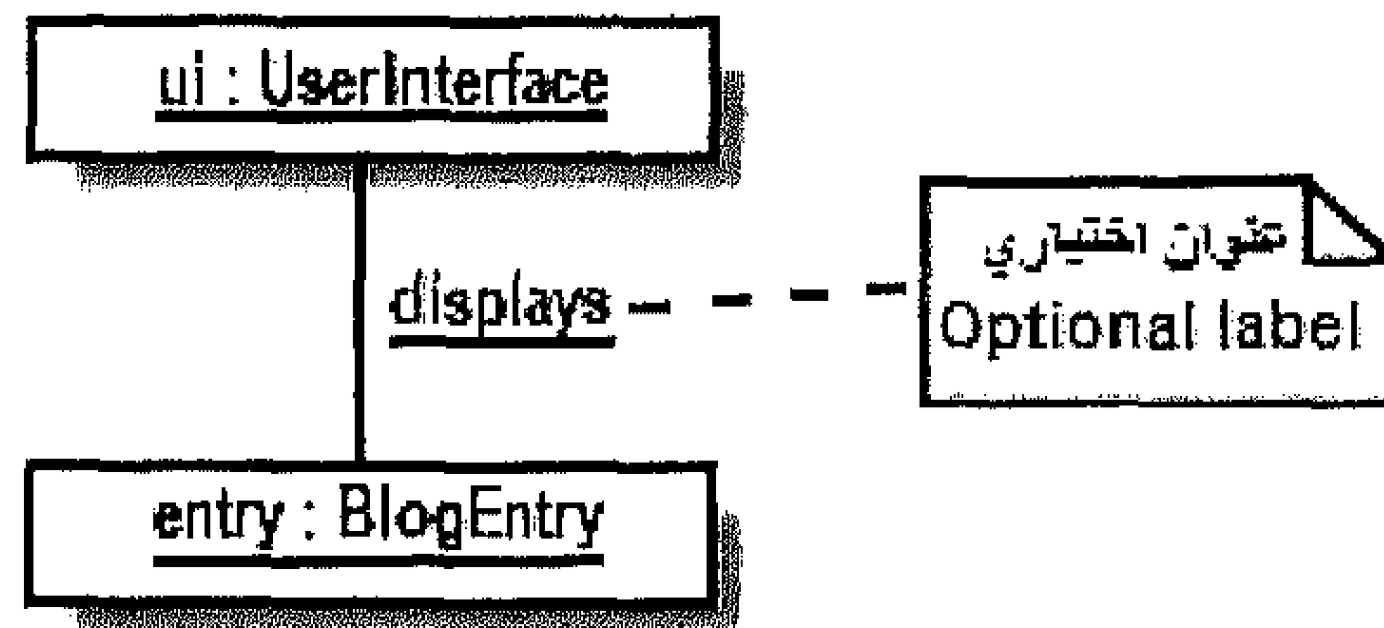


شكل رقم (٦-٥) تم استعمال خط لإظهار رابط بين كائنَين مرتبطَين.

في مخطط الكائنات، تظهر الروابط بين الكائنات إمكانية تواصلها فيما بينهما. غير أننا لا نستطيع الربط بين الكائنات لمجرد

الربط. إذا أنشأنا رابطاً بين كائنَين، فلا بد من وجود علاقة شراكة موازية بين أصنافها.

تعمل الروابط في مخطط الكائنات بطريقة مشابهة للروابط التي في مخطط الاتصال (انظر إلى الفصل الثامن). على أية حال، بخلاف مخططات الاتصال، يمكن بشكل اختياري إضافة معلومة واحدة للرابط كعنوان له تشير إلى الهدف منه، كما هو معروض في الشكل رقم (٦-٦).

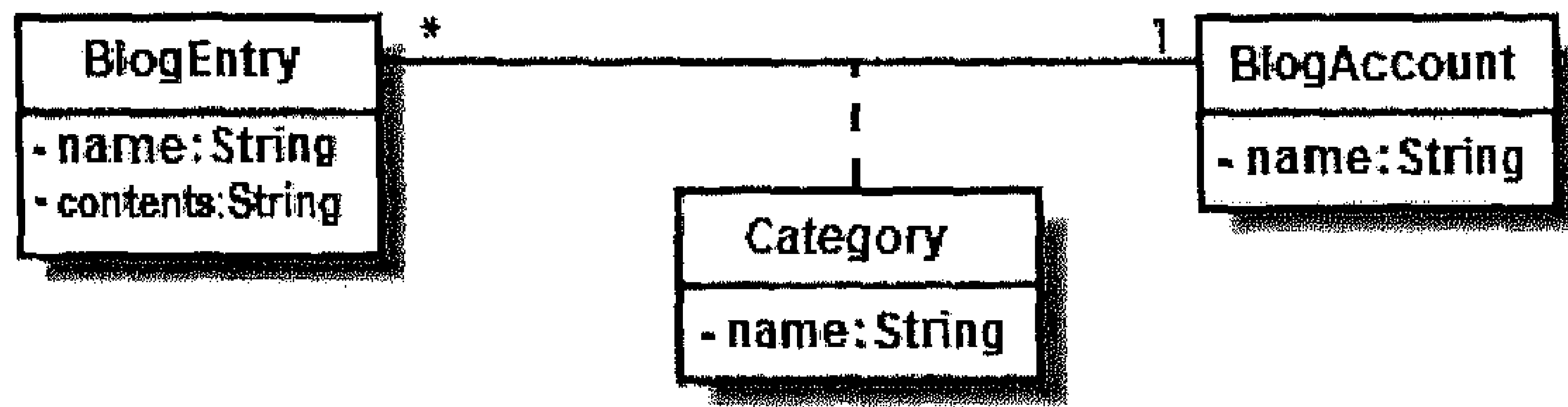


شكل رقم (٦-٦) يتم ربط كائن **BlogEntry** بكائن **UserInterface** لعرض التدوينة.

٦-٢-١ الروابط و القيود Links and Constraints

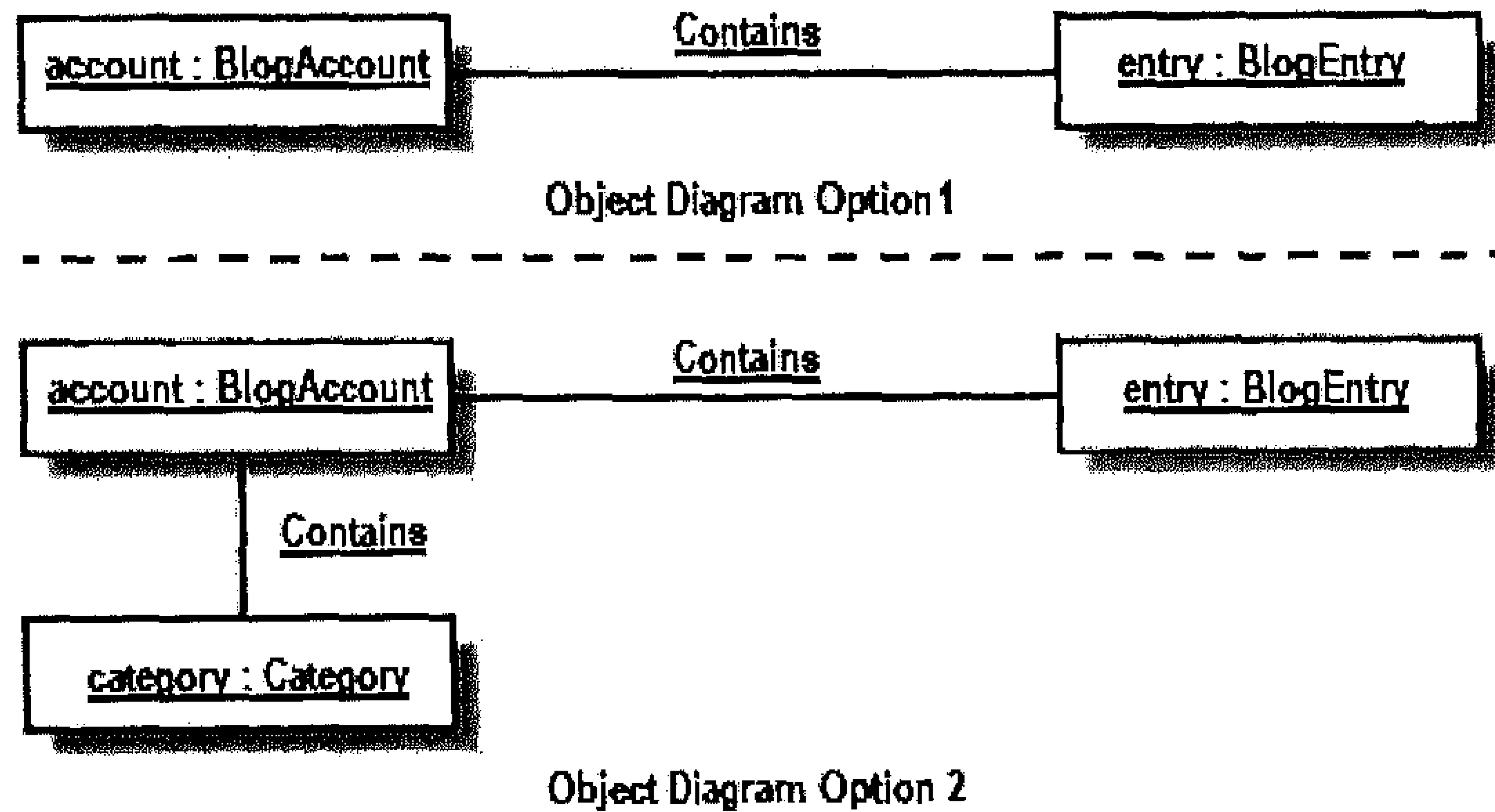
توازي الروابط التي بين الكائنات الشراكات التي بين أصنافها. وهذا يعني أنه يجب على هذه الروابط الإبقاء على قواعد القيود المطبقة على الشراكات الموازية لها.

وفي الفصل الخامس، تم نمذجة العلاقة بين BlogEntry، BlogAccount، و Category باستعمال صنف شراكة، كما هو معروض في الشكل رقم (٦-٧).



شكل رقم (٦-٧) عند إضافة BlogEntry إلى BlogAccount، سيتم تجميعها تحت فئة واحدة أو أكثر؛ يشترك الصنف Category بالعلاقة بين BlogEntry وBlogAccount.

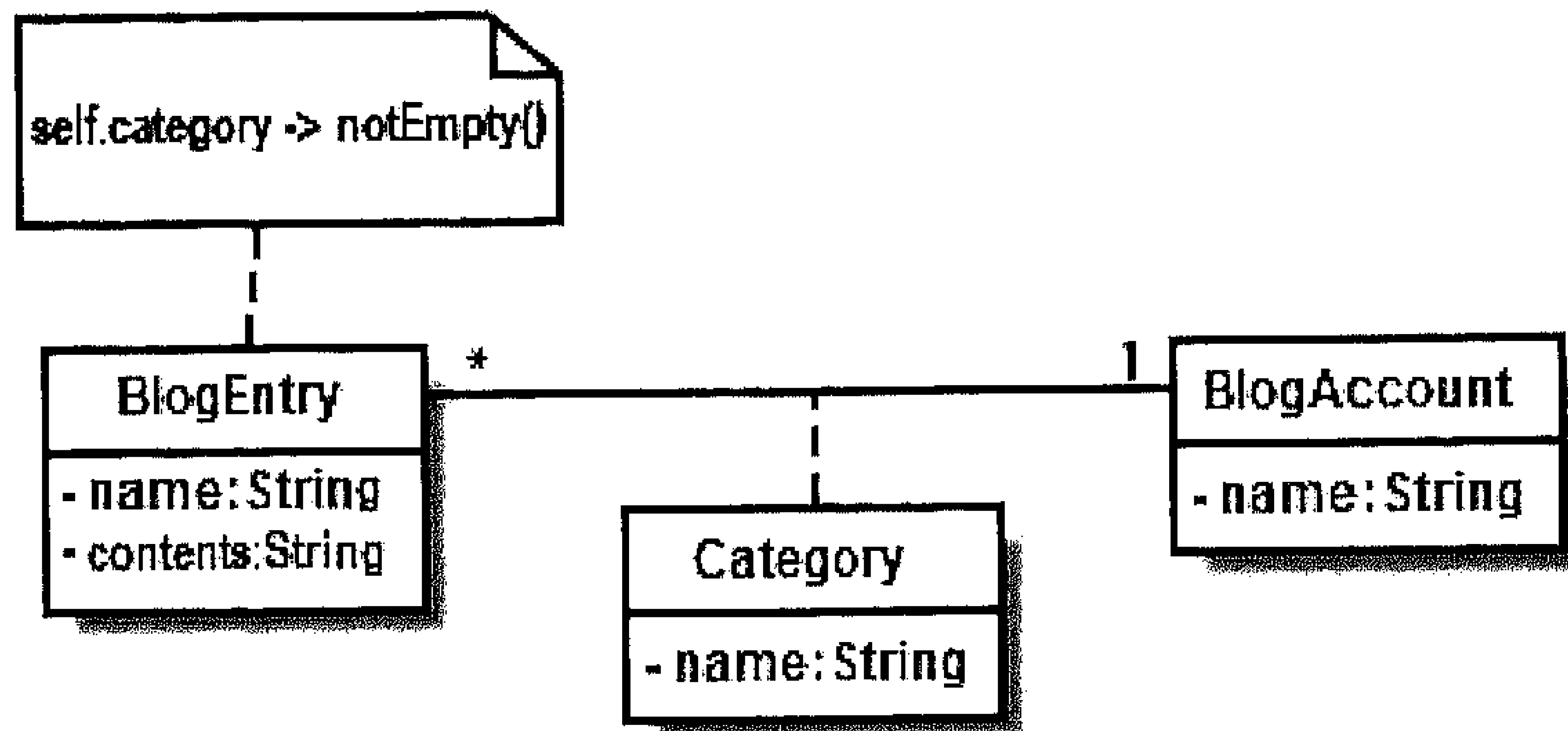
إذا تم ترك المخطط الذي في الشكل رقم (٦-٧) على حاله، سيكون مخططا الكائنات الظاهرين في الشكل رقم (٦-٨) صحيحَين تماماً.



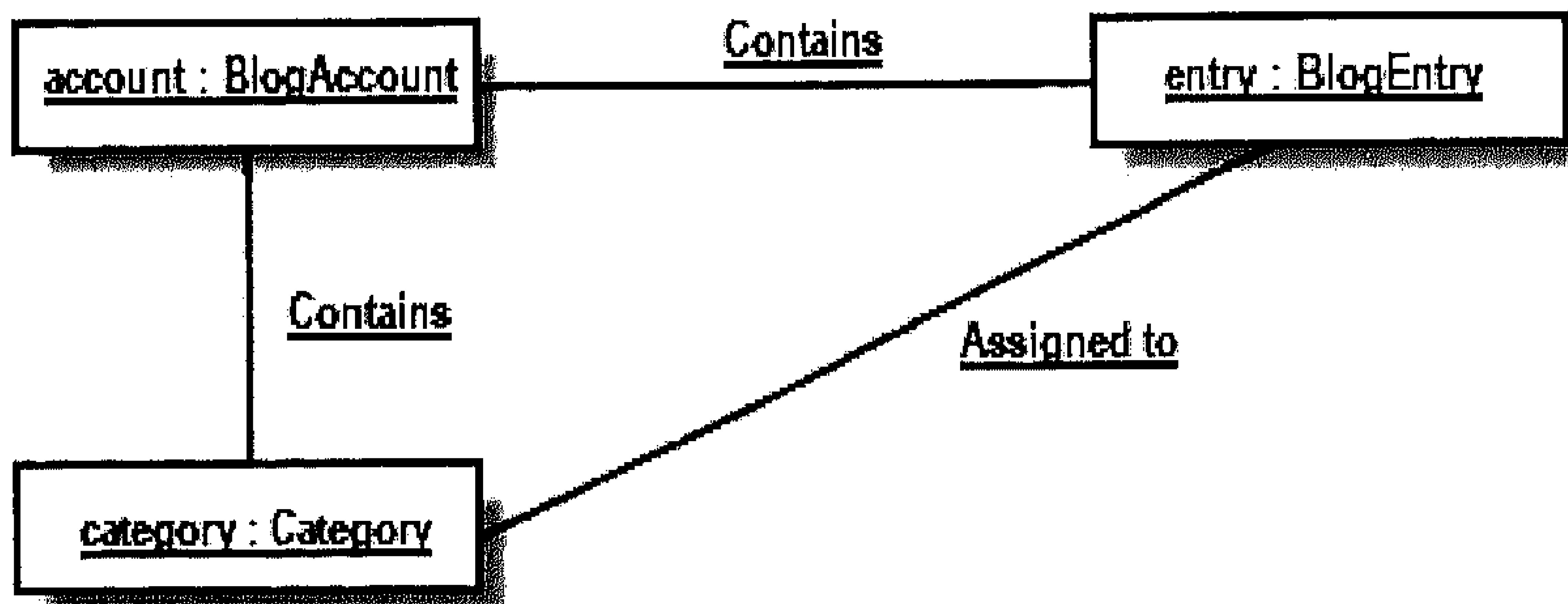
شكل رقم (٦-٨) يمكن أن يشترك BlogEntry مع BlogAccount ومع مجموعة فئات categories، لكن لا توجد قاعدة تفرض مشاركة BlogEntry مع Category.

إذا أردنا إظهار وجوب اشتراك تدوينة ما مع فئة Category معينة عند احتواء حساب المدونة لتلك التدوينة، نحتاج إلى إضافة بعض من لغة قيود الكائن OCL إلى مخطط الصنف الأصلي، كما هو معروض في الشكل رقم (٦-٩). ولقد تم تغطية OCL بإيجاز في الفصل الخامس، و تم تغطيتها بتعمق أكثر في الملحق أ.

ويجب على الكائنات وروابطها الالتزام بالقواعد الموضوعة من قبل أوامر OCL - هذا أحد أسباب تسمية OCL بلغة قيود الكائن Object Constraint Language وليس لغة قيود الصنف. مع القيود المطبقة على مخطط الأصناف في الشكل رقم (٦-٩)، تم تقليص الخيارات المحتملة لمخطط الكائنات بالارتكاز على هذه الأصناف إلى شيء ذات معنى أكثر بكثير، كما هو معروض في الشكل رقم (٦-١٠).



شكل رقم (٦-٩) يصرح القيد ضمن سياق BlogEntry، أنه يجب على كائن BlogEntry الاشتراك فعليا مع فئة category، أي يجب أن لا تكون هذه الشراكة null؛ كي تتم إضافة BlogEntry إلى BlogAccount، يجب تخصيص فئة category لها.



شكل رقم (٦-١٠) يؤثر القيد ()self.category->notEmpty على الخيارات المتوفرة عند إنشاء مخطط الكائنات، وذلك لضمان أن التدوينة تشترك مع فئة category خاصة بها كي تتم إضافتها إلى BlogAccount.

٦-٣ ربط الأصناف القوالب
Binding Class Templates

لقد رأينا في الفصل الخامس كيفية إنجاز بارامترات الأصناف القوالب باستعمال الأصناف الفرعية في مخطط الأصناف. ورغم عمل هذا الأسلوب بشكل جيد، تظهر القوة الحقيقية للقوالب في ربط بارامتراتها عند وقت التشغيل. لإجراء ذلك، يمكنك إخبار القالب بأنواع بارامتراته عند إنشاء كائن منه.

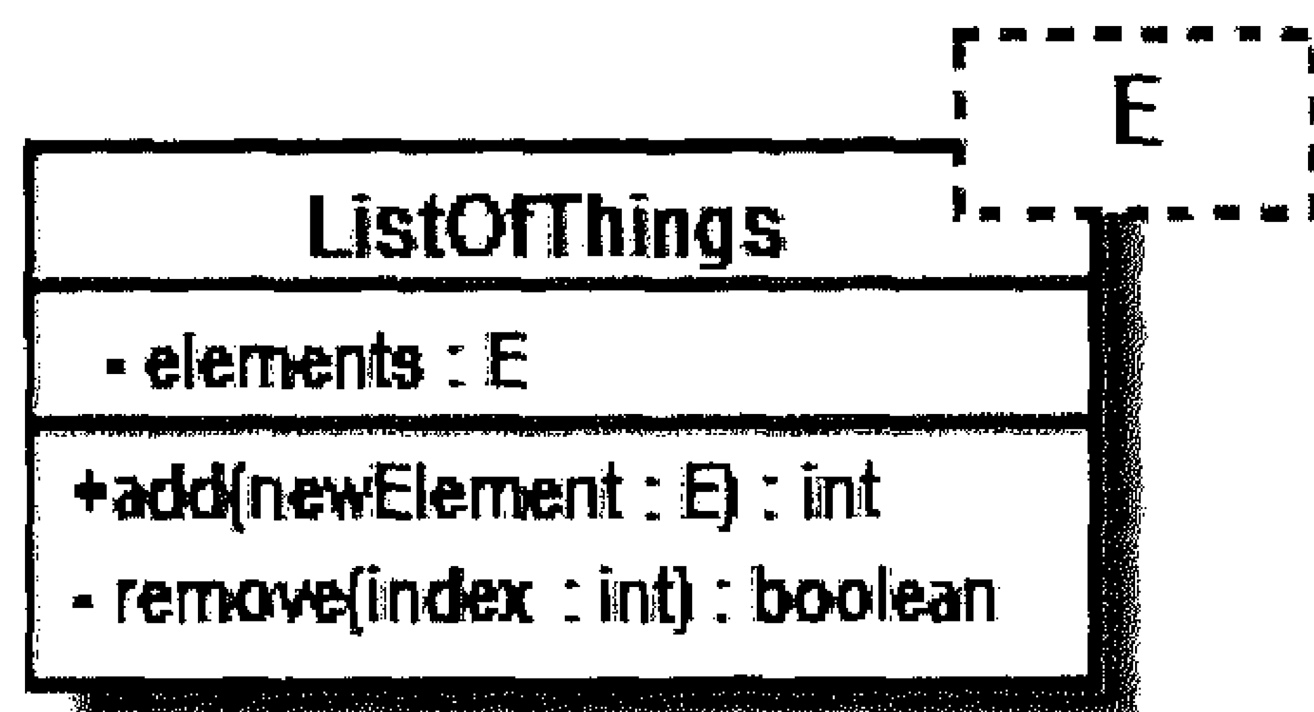
تكون مخططات الكائنات مثالية لنمذجة كيفية حدوث الربط وقت التشغيل. وعند استعمال ربط بارامتر القالب وقت التشغيل، نكون نتحدث في الحقيقة عن الكائنات و ليس عن الأصناف، لذلك لا يمكن نمذجة هذه المعلومات في مخطط أصناف عادي.

ويعرض الشكل رقم (٦-١١) صنف قالب بسيط لمجموعة عبارة عن قائمة أخذت من الفصل الرابع. تعتبر المجموعات مرشّحاً بارزاً للقوالب؛ لأنها تحتاج إلى إدارة مجموعة كائنات من دون الاهتمام بأصنافها.

ولنمذجة عملية ربط البارامتر E الخاص بالقالب ListOfThings بصنف محدد عند وقت التشغيل، كل ما عليك فعله هو إضافة تفاصيل ربط البارامتر إلى نهاية توصيف صنف الكائن، كما هو معروض في الشكل رقم (٦-١٢).



رغم عرض هذا الكتاب للغة النمذجة الموحدة من ناحية أنواع المخططات، فإن مواصفات لغة النمذجة الموحدة لا تتقيد فعلياً بمجموعة محددة من المخططات. في الحقيقة، ونستطيع عرض ترميز مخطط الكائنات على مخطط الأصناف، إذا أردنا تجميع الأصناف و ربطها وقت التشغيل في نفس المخطط.



شكل رقم (٦-١١) يمكن للمجموعة ListOfThings أن تخزن و تحذف أي صنف لكائن مرتبط به البارامتر E.

listOfBlogEntries : ListOfThings <E -> BlogEntry>

شكل رقم (٦-١٢) يقوم listOfBlogEntries بإعادة استعمال القالب العام ListOfThings، و ذلك بربط البارامتر E بالصنف BlogEntry لتخزين كائنات منه فقط.

إلى وقت قريب، كان من المستحيل عرض ربط القوالب وقت التشغيل في جافا؛ بسبب عدم دعمها للقوالب أصلاً. وعلى أية حال، مع الإصدار ٥ للغة جافا و ميزة التعميم الجديدة فيها، أصبح بالإمكان الآن بلغة جافا برمجة ربط الصنف BlogEntry وقت التشغيل بالبارامتر E التابع للقالب ListOfThings، كما هو معروض في المثال رقم (٦-٢).

توجد أمور كثيرة عن التعميم generics في جافا؛ حيث عرضنا هنا عملية إنجاز قالب بسيط فقط. للحصول على أمثلة إضافية عنها يمكن الرجوع للكتاب Java 5 Tiger: A Developer's Notebook (O'Reilly).

مثال رقم (٦-٢) استعمال خاصية التعميم بالإصدار ٥ لجافا لإنجاز القالب **ListOfThings** وربطه وقت تشغيل لينتج قائمة تدوينات **ListOfBlogEntries**.

```
public class ListOfThings<E> {
 // ListOfThings يوجد فعلياً لدى جافا قالب قائمة واستعملنا بالتالي عملياته لأجل الصنف
 private List[E] elements;
   public ListOfThings {
     elements = new ArrayList<E>( );
   }
   public int add(E object) {
     return elements.add(object);
   }
   public E remove(int index) {
     return elements.remove(index);
   }
}
 public class Application {
   public static void main(String[] args) {
   // ربط بارامتر القالب بالصنف تدوينة لتصبح القائمة تخزن كائنات تدوينات فقط
     ListOfThings<BlogEntry> listOfBlogEntries = new ListOfThings<BlogEntry>( );
   }
}
```

٦-٤ ما هي الخطوة التالية؟

بعد الأخذ بالاعتبار ميزات وقت التشغيل للنظام، من الطبيعي المتابعة بهذا النهج من خلال دراسة مخططات التتابع وخططات الاتصال. وتعرض هذه المخططات الرسائل الممررة بين الأجزاء في النظام وتظهر بذلك كيفية استعمال الكائنات.

ويمكن أن تجد مخططات التتابع في الفصل السابع؛ وستتم تغطية مخططات الاتصال في الفصل الثامن.

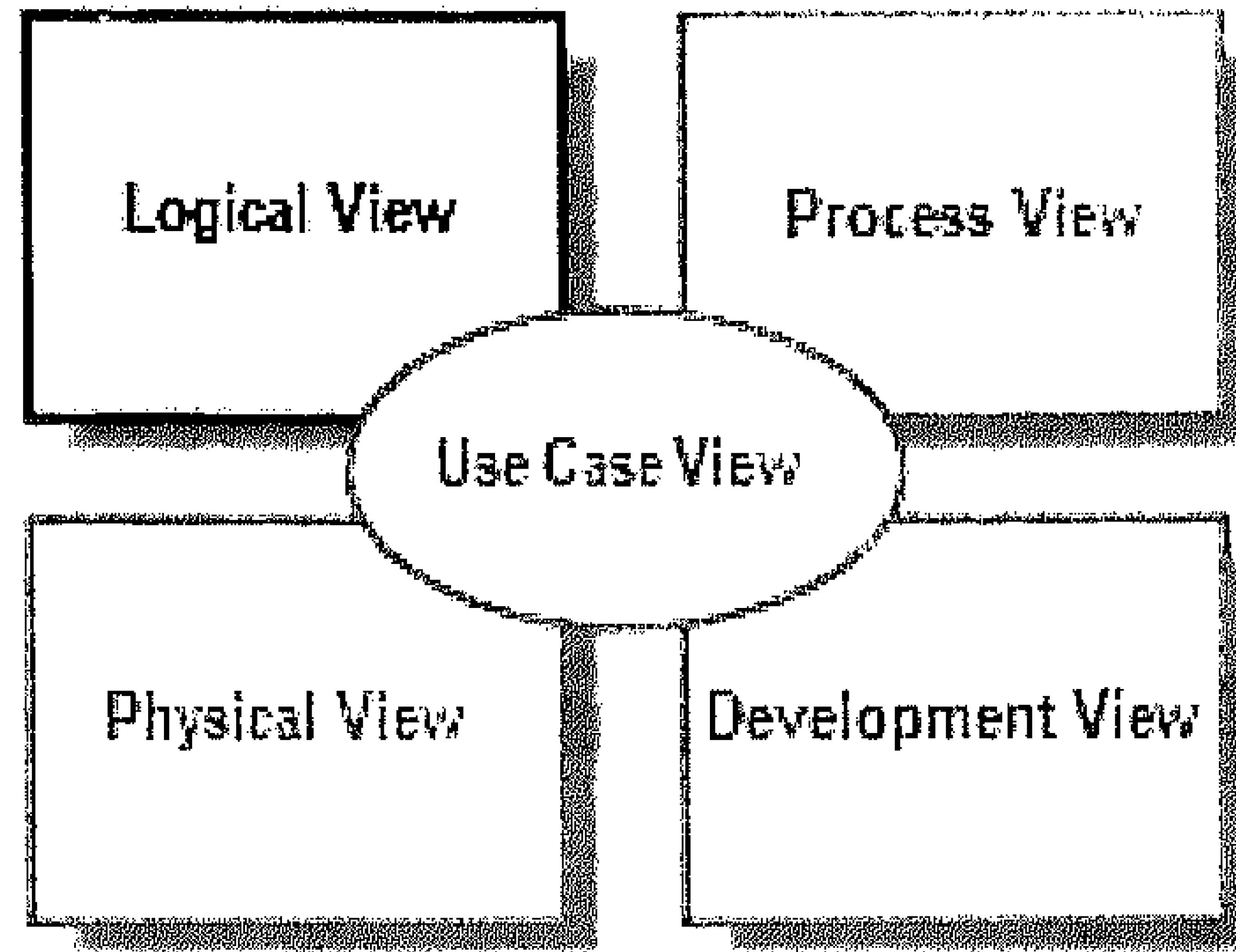
نمذجة التفاعلات المرتبة: مخططات التتابع

MODELING ORDERED INTERACTIONS: SEQUENCE DIAGRAMS

تسمح حالات الاستخدام للنموذج بوصف ما يجب أن يكون النظام قادراً على عمله؛ وتسمح الأصناف للنموذج بوصف الأنواع المختلفة للأجزاء المؤلفة لبنية النظام. هناك جزء كبير غائب عن هذه المعضلة؛ مع حالات الاستخدام والأصناف وحدهما لا نستطيع نمذجة "كيف" سيقوم النظام فعلياً بعمله. من هنا يأتي دور مخططات التفاعل و بشكل خاص مخططات التتابع.

وتشكل مخططات التتابع عنصراً مهما في المجموعة المعروفة بمخططات التفاعل. وتنمذج مخططات التفاعل تفاعلات وقت التشغيل المهمة بين الأجزاء المؤلفة للنظام، و تشكل جزءاً من المنظور المنطقي للنموذج، كما هو معروض في الشكل رقم (٧-١).

ليست مخططات التتابع وحيدة في هذه المجموعة؛ لكنها تعمل إلى جانب مخططات الاتصال (انظر إلى الفصل الثامن) ومخططات التوقيت (انظر إلى الفصل التاسع) للمساعدة في النمذجة الدقيقة لكيفية تفاعل الأجزاء المؤلفة للنظام.



شكل رقم (٧-١) يحتوي المنظور المنطقي للنموذج على التوصيفات المجردة لأجزاء النظام، بما في ذلك التفاعلات بين تلك الأجزاء.



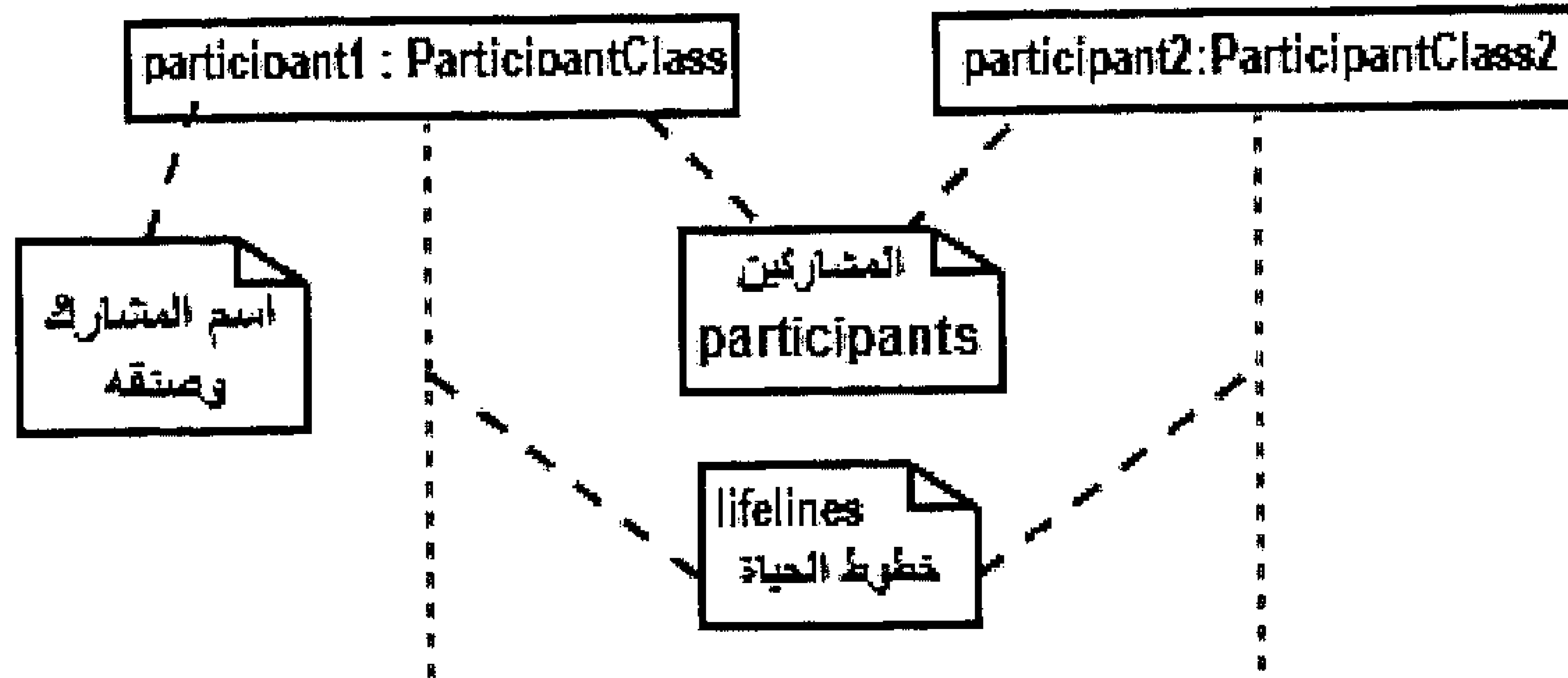
إن مخططات التتابع هي الأكثر شعبية من بين الأنواع الثلاثة لمخططات التفاعل. ربما هذا بسبب عرضها للأنواع الصحيحة من المعلومات، أو ببساطة بسبب ميلها لتكون مفهومة للناس الجدد في تعلم لغة النمذجة الموحدة.

تدور مخططات التتابع كلها حول أسر ترتيب التفاعلات بين أجزاء النظام. وباستعمال مخطط التتابع، يمكن وصف أي تفاعلات سيتم إطلاقها triggered عند تنفيذ حالة استخدام محددة، وبأي ترتيب ستحدث تلك التفاعلات. تظهر مخططات التتابع كثيراً من المعلومات الأخرى حول تفاعل ما، لكن تكمن براعتها في الأسلوب البسيط والفعّال الذي من خلاله تبين ترتيب الأحداث داخل التفاعل.

٧-١ المشاركون في مخطط التتابع
Participants in a Sequence Diagram

يتألف مخطط التتابع من مجموعة مشاركين تمثل أجزاء النظام المتفاعلة فيما بينها خلال التتابع. ومن المهم اختيار المكان المناسب

للمشاركين على مخطط التتابع. بغض النظر عن المكان الذي سيوضع فيه المشارك بشكل عمودي، يتم ترتيب المشاركين دائماً عند نفس المستوى الأفقي بدون تداخل مشاركَين مع بعضهما، كما هو معروض في الشكل رقم (٧-٢).



شكل رقم (٧-٢) يتألف أبسط مخطط تتابع من مشارك واحد فأكثر. و من الغريب جداً وجود مشارك واحد فقط في مخطط التتابع لكن ذلك قانوني تماماً في UML.

لدى كل مشارك خط حياة موازياً له ينتهي بأسفل الصفحة. يعبّر خط حياة المشارك ببساطة عن وجود الجزء عند تلك النقطة في التتابع، وهذا الخط في الحقيقة مهم فقط عند إنشاء جزء أو حذفه أثناء التتابع (انظر إلى "إنشاء الرسائل وحذفها من قبل مشارك" لاحقاً في هذا الفصل).

٧-١-١ أسماء المشاركين Participant Names

يمكن تسمية المشاركين في مخطط التتابع بعدة أساليب مختلفة، ويتم ذلك بتحديد العناصر من الصيغة القياسية التالية:

Name [selector]: class_name **ref** decomposition

وتعتمد عناصر الصيغة التي تختار استعمالها لأجل مشارك ما على المعلومات المعروفة حول المشارك في وقت محدد، كما هو مبين في الجدول رقم (٧-١).

جدول رقم (٧-١) كيف علينا فهم مكونات اسم مشارك.

مثال عن اسم لمشارك	التوصيف
Admin	تم تسمية جزء ما مدير admin، ولكن لم يتم تعيين صنف هذا الجزء عند هذه النقطة من الوقت.
: ContentManagementSystem	المشارك من الصنف ContentManagementSystem، ولكن ليس للجزء اسم خاص به حاليا.
admin: Administrator	يوجد جزء لديه الاسم admin وهو من الصنف Administration.
eventHandlers[2]: EventHandler	يوجد جزء يتم الوصول إليه داخل مصفوفة عند العنصر ٢، وهو من الصنف EventHandler.
: ContentManagementSystem ref cmsInteraction	المشارك من الصنف ContentManagementSystem، ويوجد مخطط تفاعل آخر اسمه cmsInteraction يعرض كيف يعمل المشارك داخلياً (انظر إلى "ملخص موجز عن أجزاء الأنواع في UML 2.0" لاحقاً في هذا الفصل.

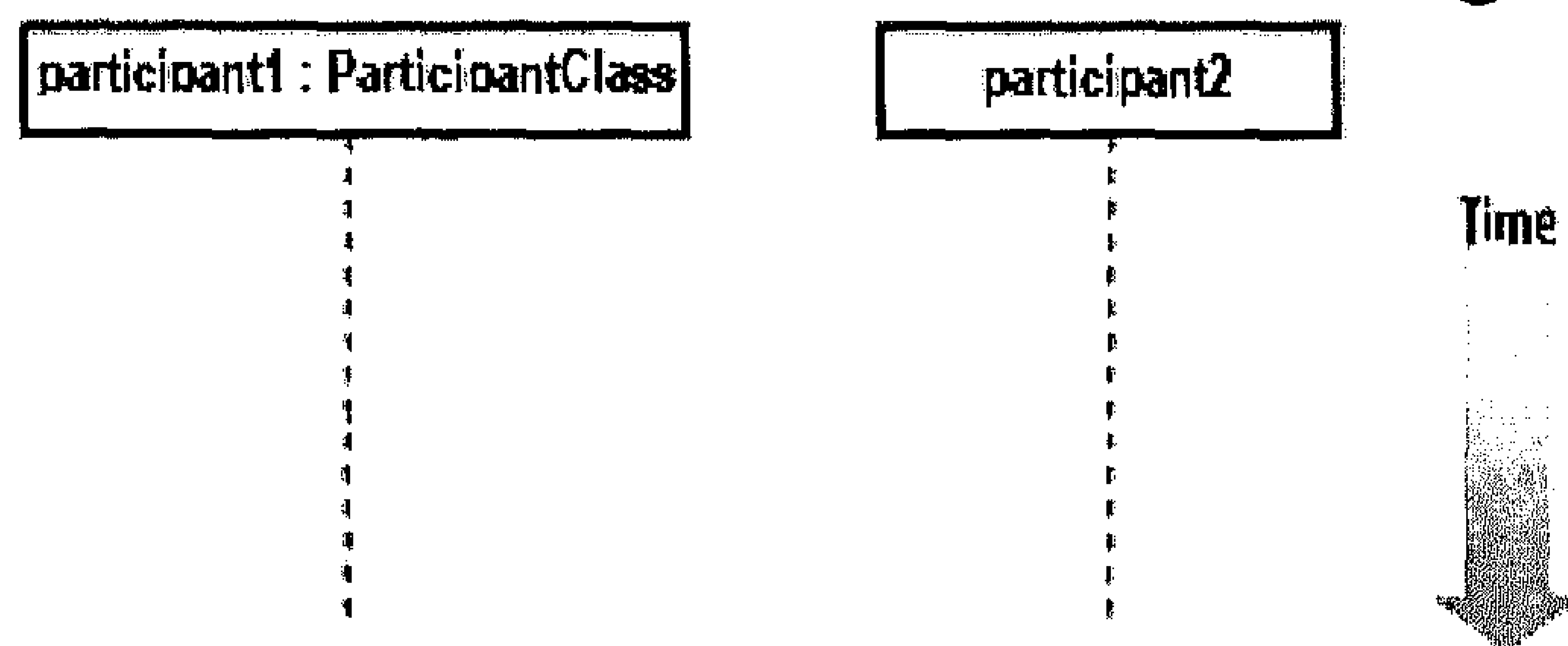
تعود الصيغة المستعملة عند إنشاء أسماء المشاركين إليك كلياً، أو ربما تعود إلى الأسلوب المعتمد في شركتك. في هذا الكتاب، نجعل أول كلمة من اسم المشارك بأحرف صغيرة لتقليل الالتباس بقدر الإمكان مع اسم الصنف. وعلى أية حال، هذا عُرف خاص بنا فقط وهو شبيه بالأعراف المستعملة عند تسمية الكائنات والأصناف في لغة جافا وهو ليس أمراً محدداً من قبل لغة النمذجة الموحدة.

ماذا عن الكائنات؟ What Happened to Objects

في UML 1.x ، عادة ما يكون المشاركون في مخطط التفاعل كائنات برمجية بالمعنى التقليدي للغات البرمجة الكائنية التوجه. ويكون كل كائن مثيلاً لصنف ما، ويتم وضع خط تحت اسم الكائن للإشارة إلى ذلك. وبما أن UML 2.0 هي أكثر من لغة نمذجة أنظمة عامة، يكون التفكير بها كأجزاء من النظام تتفاعل فيما بينها ذات معنى أكثر بكثير من التفكير بها ككائنات برمجية. لهذا السبب قمنا باستعمال التعبير "مشارك" لوصف جزءٍ متعلق بالتفاعلات في مخطط التتابع. وما زال بالإمكان أن يكون المشارك كائناً برمجياً، مثل أسلوب UML 1.x، لكن يمكن أن يكون أيضاً أي جزء آخر من النظام، وذلك ليتماشى مع UML 2.0.

٧-٢ الوقت Time

يصف مخطط التتابع الترتيب الذي تحدث فيه التفاعلات، لذلك يعتبر الوقت عاملاً مهماً فيه. ويعرض الشكل رقم (٧-٣) العلاقة بين مخطط التتابع والوقت.



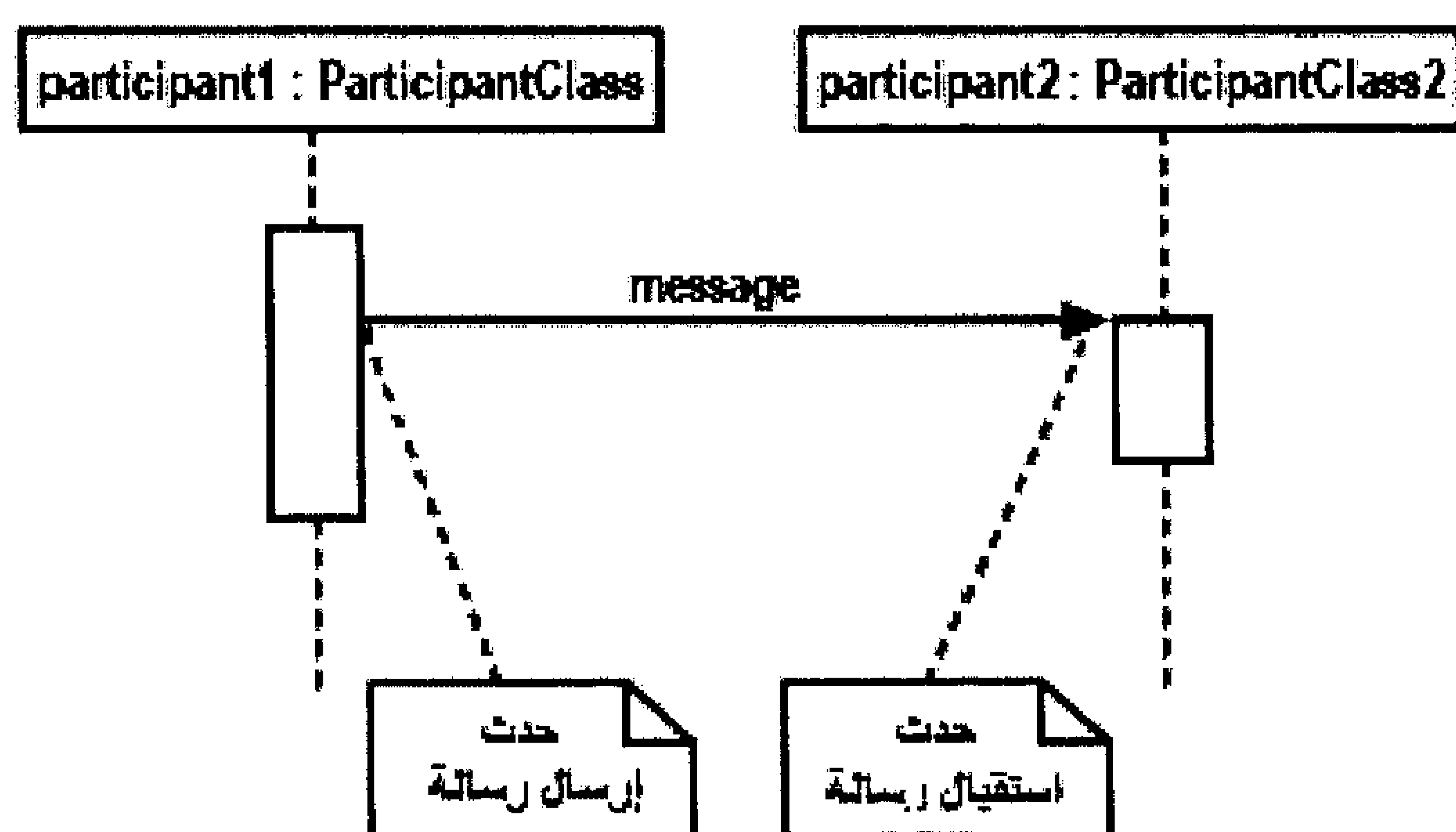
شكل رقم (٧-٣) اتجاه الوقت نحو أسفل الصفحة في مخطط التتابع بالتوافق مع خط حياة المشارك.

يبدأ الوقت على مخطط التتابع عند أعلى الصفحة تحت عنوان أعلى مشارك بالضبط، ويتجه بعد ذلك لأسفل الصفحة. ويشير ترتيب وضع التفاعلات على مخطط التتابع إلى الترتيب الذي تحدث به تلك التفاعلات بمرور الوقت.

ويتمحور الوقت على مخطط التتابع حول الترتيب الزمني وليس حول المدة الزمنية. وبالرغم من الإشارة إلى وقت حدوث أي تفاعل في مخطط التتابع من خلال المكان الموضوع فيه عمودياً في المخطط، ليس للكم الذي سيأخذه التفاعل من الحيز العمودي علاقة بالمدة الزمنية التي سيأخذها التفاعل. ويهتم مخطط التتابع أولاً بأمر ترتيب التفاعلات بين المشاركين؛ سيتم عرض معلومات أكثر تفصيلاً عن التوقيت بشكل أفضل على مخططات التوقيت (انظر إلى الفصل التاسع).

٧-٣ الأحداث، والإشارات، والرسائل
Events, Signals, and Messages

إن الحدث هو الجزء الأصغر من التفاعل. والحدث هو أي نقطة في التفاعل يحدث عندها شيء ما، كما هو معروض في الشكل رقم (٧-٤).



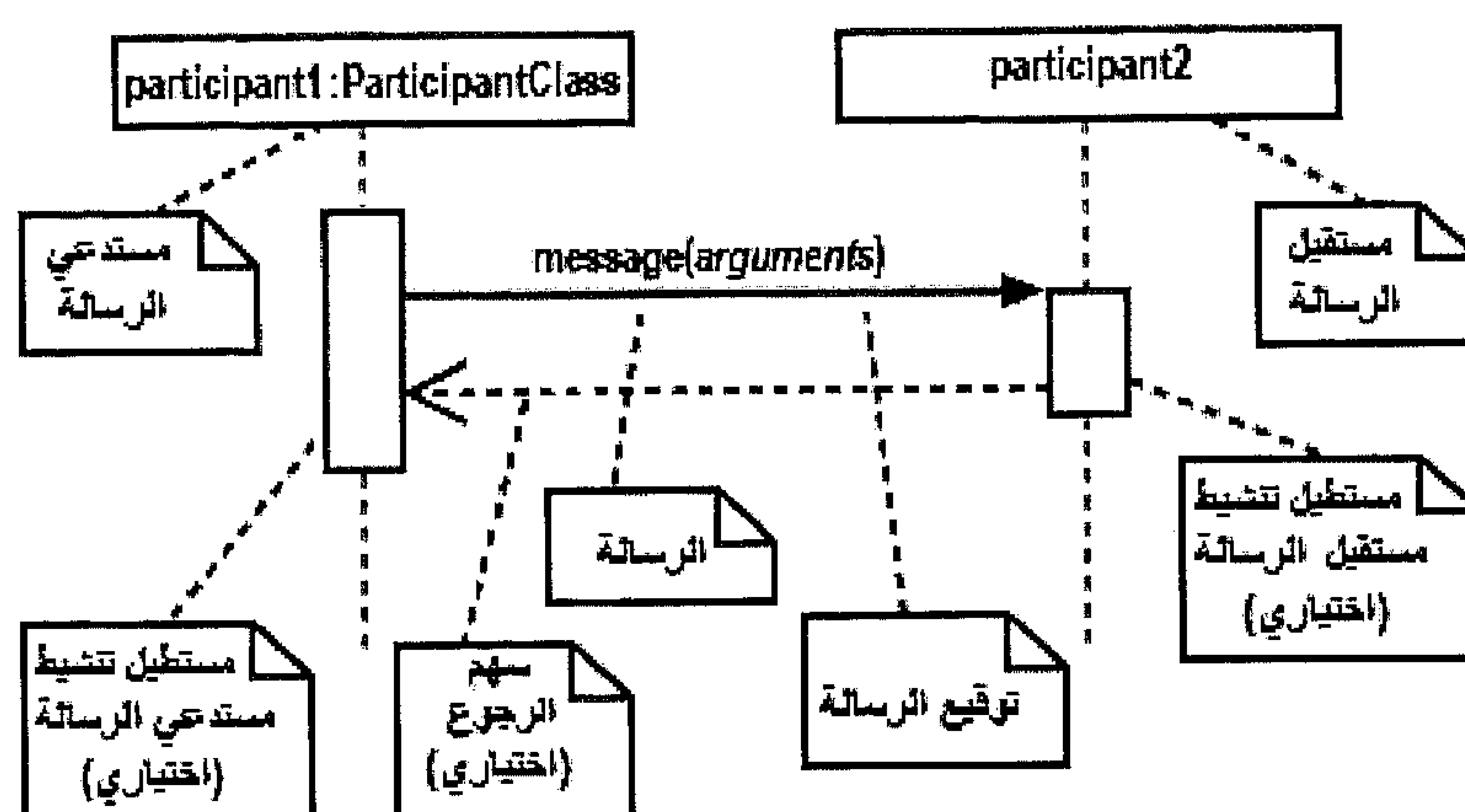
شكل رقم (٧-٤) ربما تكون الأمثلة الأكثر شيوعا عن الأحداث عند إرسال أو استقبال رسالة أو إشارة.

إن الأحداث هي الأجزاء الأساسية للإشارات والرسائل. وتعد الإشارات والرسائل في الحقيقة أسماء مختلفة لنفس المفهوم؛ فالإشارة هي

مصطلح شائع الاستعمال من قبل مصممي النظام، بينما يفضل مصممو البرامج استعمال مصطلح الرسائل.

تتصرف وتظهر الإشارات والرسائل بنفس الطريقة فيما يتعلق بمخططات التتابع، لذلك سنلتزم باستعمال مصطلح "الرسائل" في هذا الكتاب.

ويحدث التفاعل في مخطط التتابع عندما يقرر مشارك ما إرسال رسالة إلى مشارك آخر، كما هو معروض في الشكل رقم (٧-٥).



شكل رقم (٧-٥) يتم عرض التفاعلات على مخطط التتابع كرسائل بين المشاركين.

يتم تحديد الرسائل على مخطط التتابع باستعمال سهم يتجه من المشارك الذي يريد تمرير الرسالة (مستدعي الرسالة) إلى المشارك الذي عليه استلام الرسالة (مستلم الرسالة). ويمكن أن تتدفق الرسائل في أي اتجاه منطقي للتفاعل المطلوب من اليسار إلى اليمين، أو من اليمين إلى اليسار، أو حتى ترجع إلى مستدعي الرسالة نفسه. فكر بالرسالة كأنها حدث يتم تمريره من مستدعي الرسالة إلى مستقبلها ليعمل شيئاً.

٧-٣-١ تواقيع الرسالة Message Signatures

يأتي سهم الرسالة مع وصف أو توقيع حيث إن صيغة توقيع الرسالة هي كالتالي:

attribute = signal_or_message_name (arguments): return_type

يمكن تحديد أي عدد من الوسيطات arguments المختلفة بخصوص الرسالة، حيث يتم التفريق بينها باستعمال رمز الفاصلة (,). وتأتي صيغة الوسيطة الواحدة كالتالي:

<name>:<class>

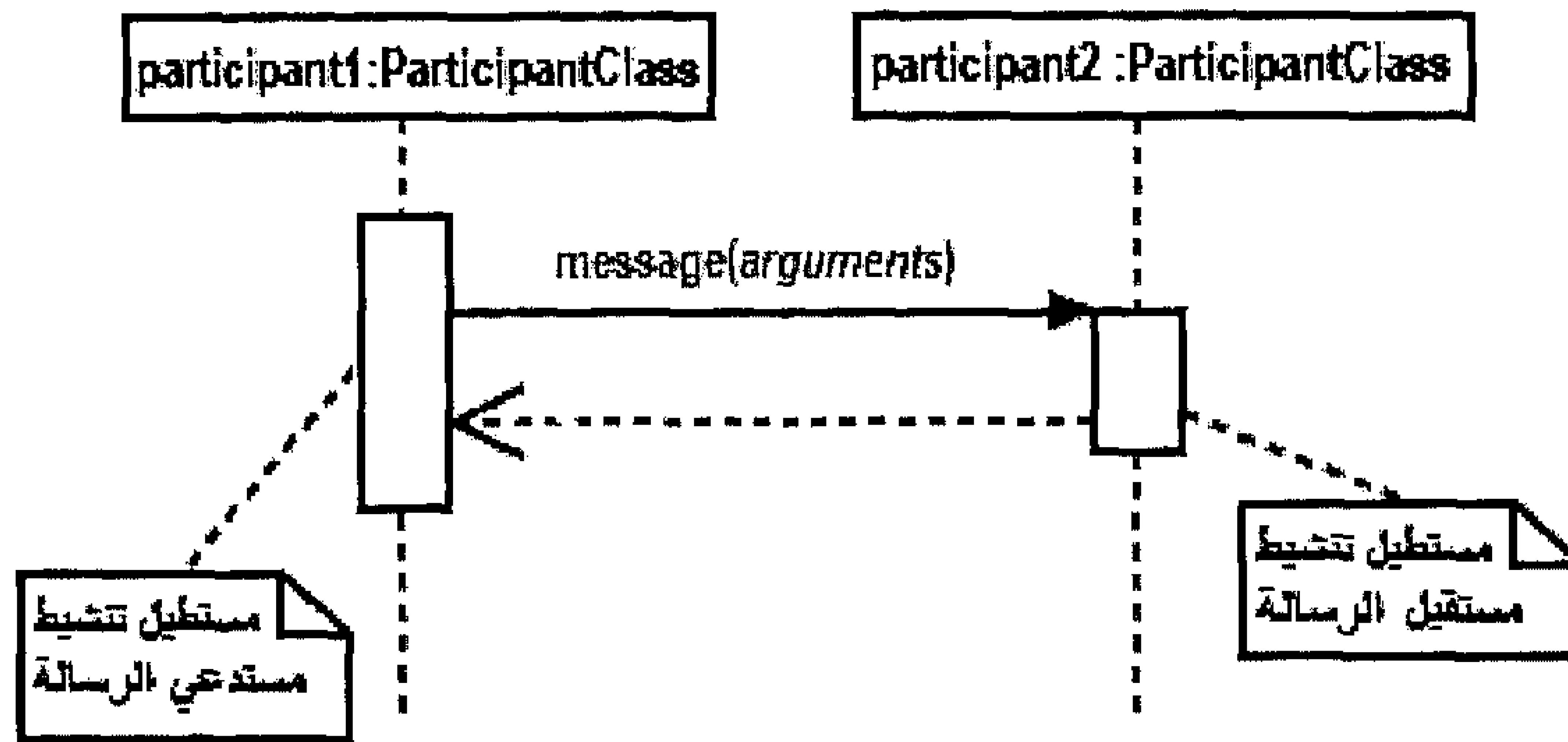
تعتمد عناصر الصيغة التي تستعملها لرسالة معينة على المعلومات المعروفة حول تلك الرسالة عند وقت محدد، كما هو مبين في الجدول رقم (٧-٢).

جدول رقم (٧-٢) كيفية فهم مكونات توقيع رسالة ما.

مثال عن توقيع رسالة	الوصف
doSomething()	اسم الرسالة هو doSomething، ولكن ليس هناك معلومات إضافية معروفة عنها.
goSomething(number1: Number, number2: Number)	اسم الرسالة هو doSomething، وهي تأخذ وسيطتين number1، و number2 من الصنف Number.
doSomething(): ReturnClass	اسم الرسالة هو doSomething، وهي لا تأخذ أية وسيطات، وترجع كائن من الصنف ReturnClass.
myVar = goSomething(): ReturnClass	اسم الرسالة هو doSomething، وهي لا تأخذ أية وسيطات، وترجع كائن من الصنف ReturnClass الذي يتم نسخه في الخاصية myVar الخاصة بمستدعي الرسالة.

٧-٤ مستطيلات التنشيط Activation Bars

عندما يتم تمرير رسالة إلى مشارك ما تقوم بإطلاق أو استدعاء المشارك المستقبل للقيام بأمر ما؛ ويقال للمشارك المستقبل عند هذه النقطة أنه يكون نشطاً **active**. ويمكن استعمال مستطيل التنشيط لإظهار المشارك أنه نشط؛ أي يعمل أمراً ما، كما هو معروض في الشكل رقم (٧-٦).



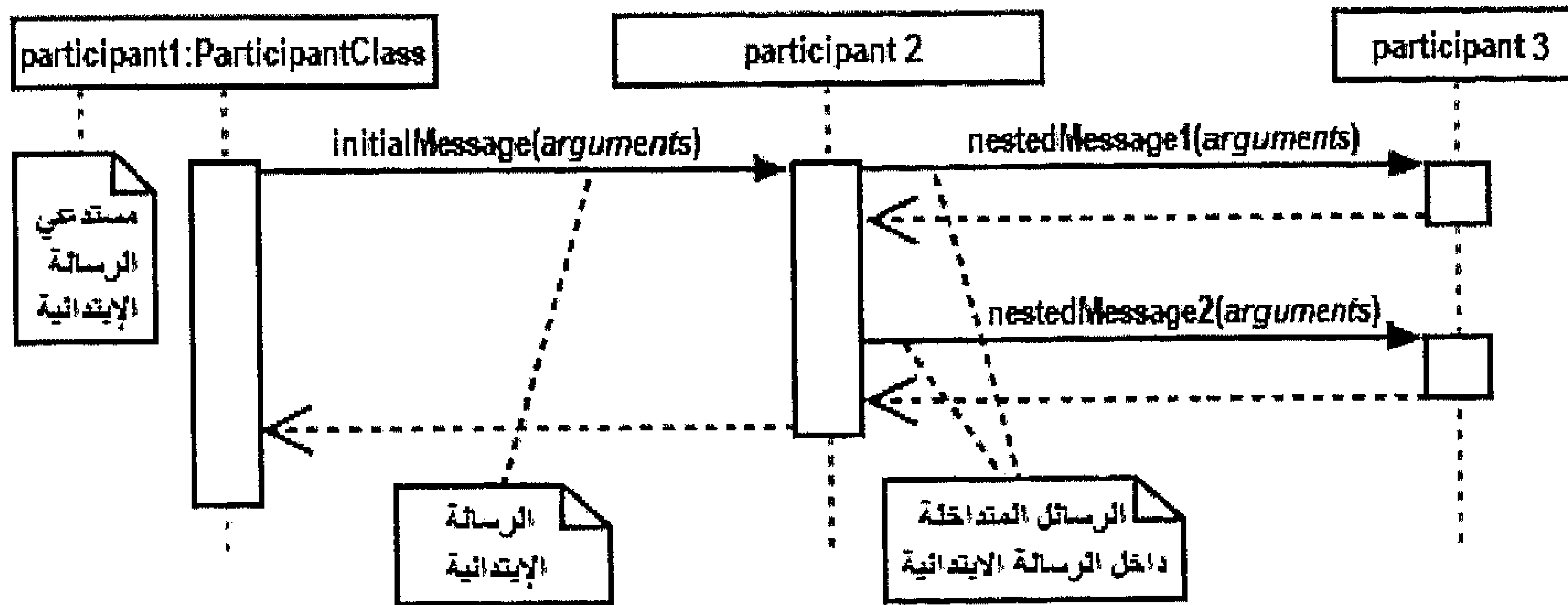
شكل رقم (٧-٦) تظهر مستطيلات التنشيط اشتغال المشارك بعمل أمر ما لفترة زمنية.

يمكن إظهار مستطيل تنشيط عند طرفي الإرسال والاستقبال للرسالة. يشير هذا إلى كون المشارك المرسل مشغولاً بينما هو يرسل الرسالة وأن المشارك المستقبل مشغولاً بعد استلامه الرسالة.

> تكون مستطيلات التنشيط اختيارية، وقد تسبب باضطراب في المخطط.

٧-٥ الرسائل المتداخلة Nested Messages

عندما تسبب رسالة من مشارك ما بإرسال رسالة أو أكثر من قبل المشارك المستقبل لها، يقال لتلك الرسائل المرسلة أنها متداخلة مع الرسالة المثارة بالبداية، كما هو معروض في الشكل رقم (٧-٧).



شكل رقم (٧-٧) استدعاء رسالتين متداخلتين حين استقبال الرسالة الابتدائية.

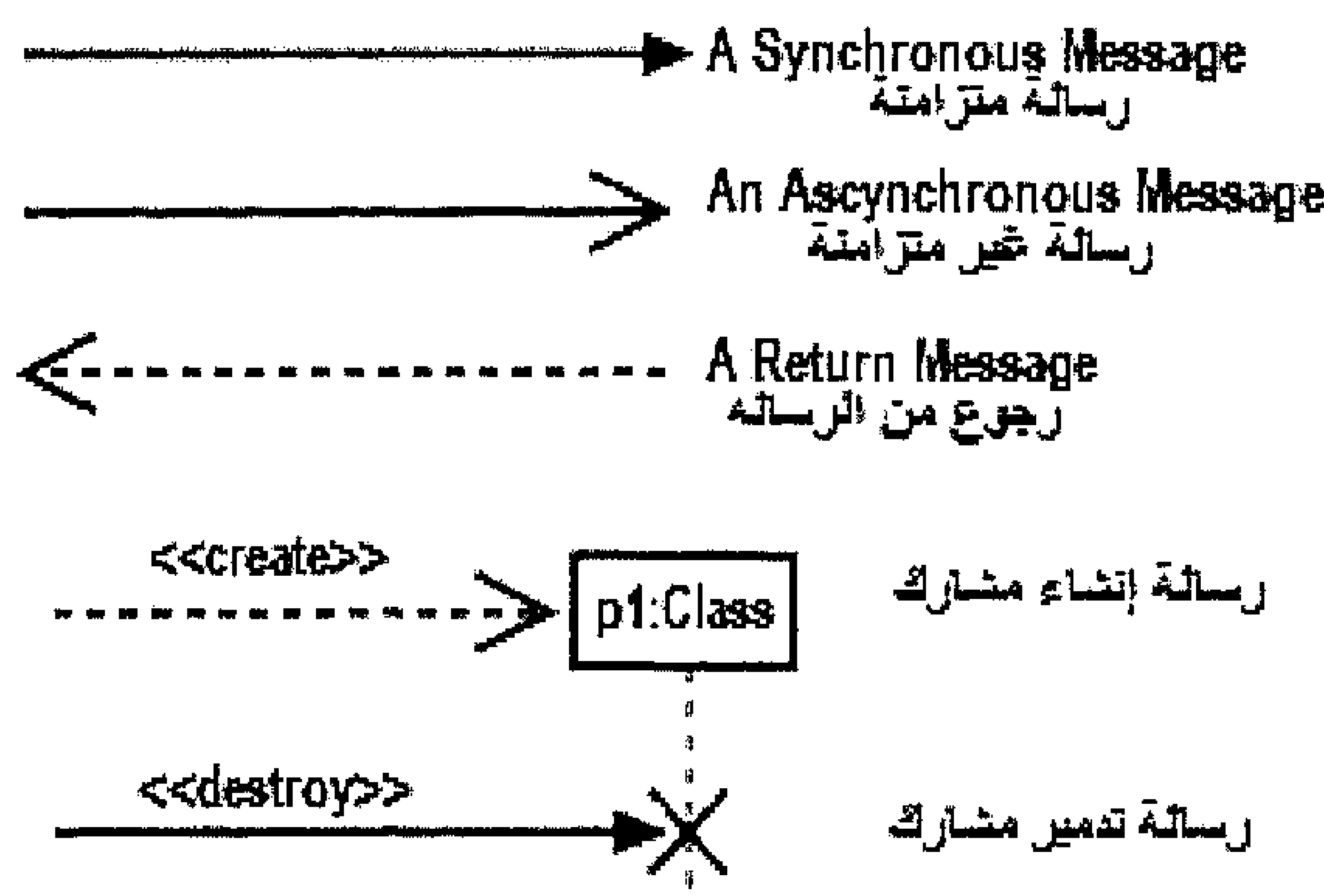
في الشكل رقم (٧-٧)، يرسل المشارك participant1 الرسالة initialMessage(..) إلى المشارك participant2. عندما يستقبل المشارك participant2 الرسالة الابتدائية initialMessage(..)، ويصبح المشارك participant2 نشطاً ويرسل رسالتين متداخلتين إلى المشارك participant3. ويمكن أن يكون عندك أي عدد من الرسائل المتداخلة داخل الرسالة المثارة، وأي عدد من مستويات الرسائل المتداخلة على مخطط التتابع.

٧-٦ أسهم الرسالة Message Arrows

إن لنوع رأس سهم الرسالة أهمية لفهم نوع الرسالة التي يتم تمريرها. على سبيل المثال، ربما يريد مستدعي الرسالة انتظار الرجوع من الرسالة قبل متابعة عمله (الرسائل المتزامنة synchronous). أو ربما يرغب فقط بإرسال الرسالة إلى مستلمها دون انتظار أي رجوع منها (الرسائل غير المتزامنة asynchronous).

وتحتاج مخططات التتابع إلى عرض هذه الأنواع من الرسائل باستعمال أسهم مختلفة لها، كما هو معروض في الشكل رقم (٧-٨).

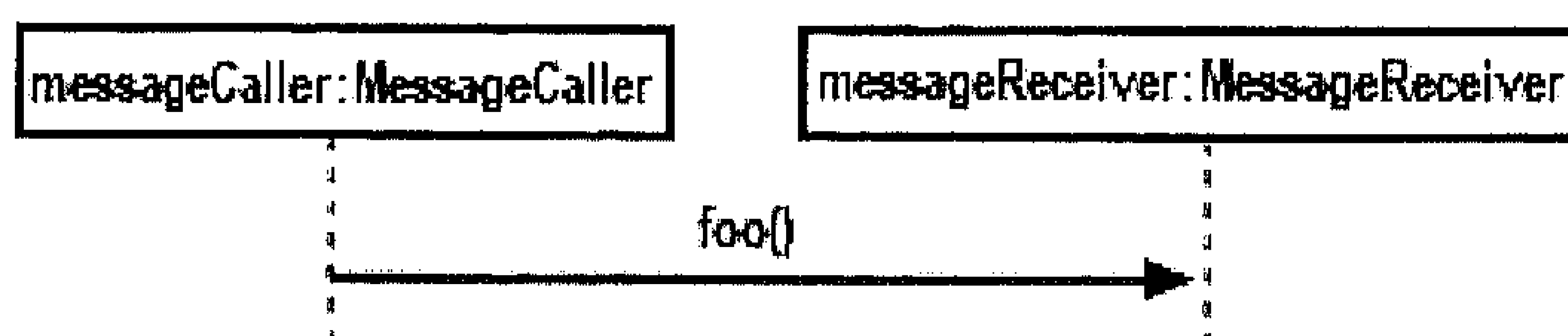
لشرح كيفية عمل هذه الأنواع المختلفة من الرسائل، دعنا ننظر إلى بعض الأمثلة البسيطة حيث يكون المشاركون في الحقيقة كائنات برمجية مبرمجة بلغة جافا.



شكل رقم (٧-٨) يوجد خمسة أنواع رئيسية من أسهم الرسائل يتم استعمالها في مخططات التتابع ولكل منها معناه الخاص.

٧-٦-١ الرسائل المتزامنة Synchronous Messages

كما هو مذكور سابقا، يتم إرسال رسالة متزامنة عندما يقوم مرسل الرسالة بانتظار الرجوع من عند مستلم الرسالة، كما هو معروض في الشكل رقم (٧-٩) وحيث تتم برمجة هذا التفاعل بلغة جافا من خلال استدعاء عادي لطريقة ما، كما هو معروض في المثال رقم (٧-١).



شكل رقم (٧-٩) يقوم المشارك messageCaller بإرسال رسالة متزامنة بسيطة إلى المشارك messageReceiver.

مثال رقم (٧-١) يقوم كائن messageCaller باستدعاء الطريقة foo() بلغة جافا لتطبيقها على الكائن messageReceiver ، و ينتظر الكائن بعد ذلك الرجوع من الاستدعاء messageReceiver. Foo() قبل متابعة تنفيذ أي خطوات أخرى في التفاعل.

```
public class MessageReceiver {
  public void foo( ) {
    قم بإجراء أي عمل داخل هذه الطريقة //
  }
}
public class MessageCaller {
  private MessageReceiver messageReceiver;
    يصرح عن الخصائص والطرق الأخرى للصنف هنا //
    في مكان آخر داخل الصنف يتم تمهيد الخاصية messageReceiver //
  public doSomething(String[] args) {
    يستدعي الكائن الحالي الطريقة foo( ) //
    this.messageReceiver.foo( ); // ثم انتظر الرجوع من الطريقة
    قبل المتابعة هنا مع ما تبقى من عمل //
  }
}
```

٧-٦-٢ الرسائل غير المتزامنة Asynchronous Messages

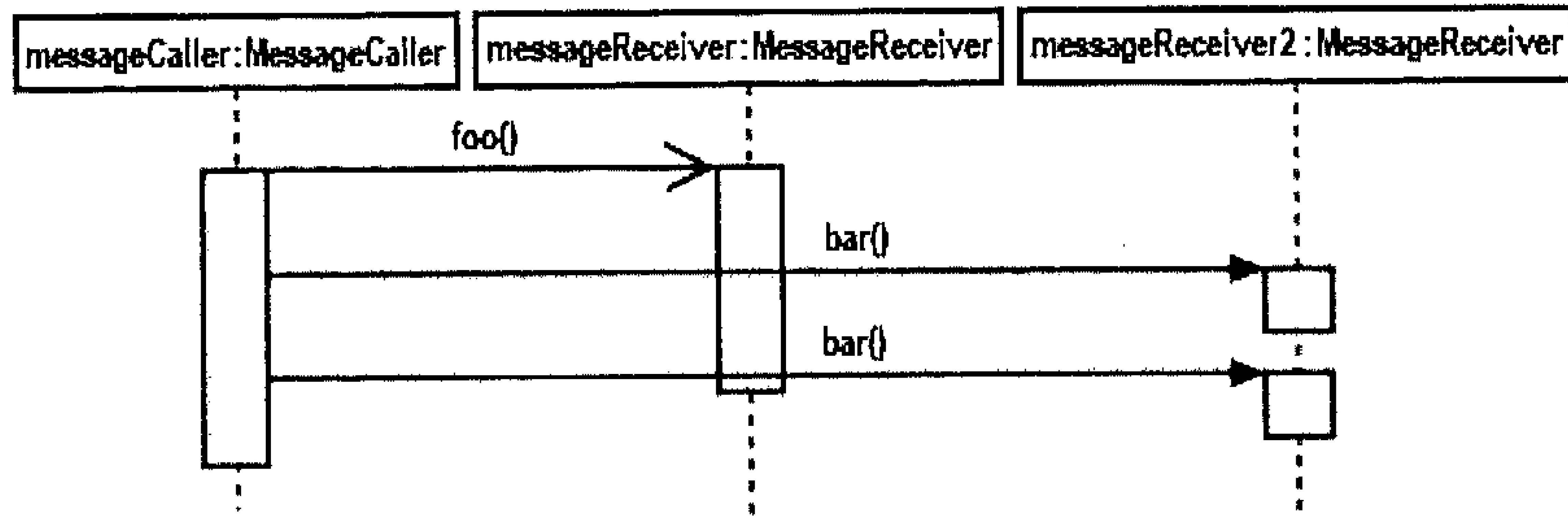
من الرائع حدوث كل تفاعلات النظام بشكل تتابعي في ترتيب بسيط و جميل. يريد كل مشارك إرسال رسالة إلى مشارك آخر ثم الانتظار بهدوء حتى الرجوع من الرسالة قبل متابعة عمله. لكن لسوء الحظ، لا تعمل أكثر الأنظمة بهذه الطريقة. ويمكن أن تحدث التفاعلات عند نفس النقطة في التوقيت، وتريد أحياناً بدء مجموعة تفاعلات بالكامل في نفس الوقت ولا تريد انتظار الرجوع منها على الإطلاق.

على سبيل المثال، لنقول أنك تصمم جزءاً من برنامج ذات واجهة مستخدم تدعم تحرير وطباعة مجموعة وثائق. يوفر تطبيقك زراً للمستخدم لطباعة وثيقة. يمكن أن تستغرق الطباعة بعض الوقت، لذلك تريد أن

تعرض أنه بعد الضغط على زر الطباعة، و خلال طباعة الوثيقة، يمكن للمستخدم أن يقوم بعمل أشياء أخرى في التطبيق. لم يعد سهم الرسالة المتزامنة العادي كافياً لعرض هذه الأنواع من التفاعلات. من هنا تحتاج إلى نوع جديد من أسهم الرسائل لاستعماله مع الرسائل غير المتزامنة.

ويتم إرسال رسالة غير متزامنة من خلال مرسل الرسالة إلى مستلم الرسالة، ولكن لا ينتظر مرسل الرسالة حتى الرجوع منها لمتابعة عمله مع باقي خطوات التفاعل. وهذا يعني أن مرسل الرسالة سيرسل رسالة إلى مستلم الرسالة وسيكون أيضاً مشغولاً بإرسال رسائل أخرى قبل الرجوع من الرسالة الأولى، كما هو معروض في الشكل رقم (٧-١٠).

وهناك طريقة شائعة لبرمجة التراسل غير المتزامن بلغة جافا من خلال استعمال المسالك threads، كما هو معروض في المثال رقم (٧-٢).



شكل رقم (٧-١٠) بينما تكون الطريقة foo() تعمل على الكائن messageReceiver، يكون الكائن messageCaller مستمراً في التفاعل بتنفيذ إرسال رسائل متزامنة أخرى إلى كائن آخر messageReceiver2.

إذا لم تكن معتاداً على كيفية عمل المسالك threads بلغة جافا، راجع الكتاب Java in a Nutshell, Fifth Edition (O'Reilly) أو الكتاب

Java Threads (O'Reilly). انظر إلى "تطبيق الرسائل غير المتزامنة" لاحقاً في هذا الفصل لرؤية مثال عملي عن الرسائل غير المتزامنة.

مثال رقم (٧-٢) تستدعي الرسالة غير المتزامنة operation1() مسلكاً داخلياً فيما يتعلق بمستلم الرسالة MessageReceiver الذي يقوم بتنشيط الرسالة (تنفيذ الطريقة run())، ثم يقوم مباشرة بإرجاع التحكم بالتنفيذ إلى الكائن messageCaller.

```
public class MessageReceiver implements Runable {
  public void operation1( ) {
    // تستقبل الرسالة و تطلق المسلك
    Thread fooWorker = new Thread(this);
    fooWorker.start();
    // يبدأ هذا الاستدعاء مسلك جديد ثم استدعاء الطريقة "رن" التي بالاسفل run
    // عند بدء المسلك بالتنفيذ ، يتم الرجوع مباشرة من استدعاء foo()
  }
  public void run( ) {
    // سيتم هنا تنفيذ العمل لأجل استدعاء الرسالة foo()
  }
}
public class MessageCaller {
  private MessageReceiver messageReceiver;
  // يصرح عن الخصائص و الطرق الأخرى للصنف هنا
  // يتم تمهيد الخاصية في مكان آخر في الصنف messageReceiver
  public void doSomething(String[] args) {
    // يستدعي الكائن الحالي الطريقة operation( )1
    this.messageReceiver.operation1( );
    // ثم يتابع مباشرة مع ما تبقى من عمل
  }
}
```

٧-٦-٣ رسالة الرجوع The Return Message

تعتبر رسالة الرجوع جزءاً اختيارياً من الترميز، ويمكن استعمالها عند نهاية مستطيل التنشيط لإظهار رجوع التحكم بمجرى التنشيط إلى

المشارك الذي أرسل الرسالة الأولى. ويشبه سهم الرجوع في شفرة البرمجة الوصول إلى نهاية الطريقة أو الوصول إلى استدعاء صريح للتعليمة return (التي تُرجع مباشرة من الطريقة).

ويمكن عدم استعمال رسائل الرجوع التي تجعل مخطط التتابع مزدحماً كثيراً و مشوّشاً. ليس عليك إفساد مخطط التتابع بسهم رجوع لكل مستطيل تنشيط، وذلك بسبب وجود سهم رجوع ضمني لكل مستطيلات التنشيط المتعلقة باستعمال الرسائل المتزامنة.



بالرغم من رواج تمرير الرسائل بين المشاركين المختلفين، فمن الطبيعي تماماً تمرير المشارك رسالة لنفسه. وتعتبر الرسائل المرسلة من كائن إلى نفسه وسيلة جيدة لتقسيم نشاط كبير إلى أجزاء أصغر وأسهل للإدارة، ويمكن التفكير فيها برمجياً بشكل مشابه جداً للقيام باستدعاء طريقة من خلال المرجع this في لغة جافا و #C.

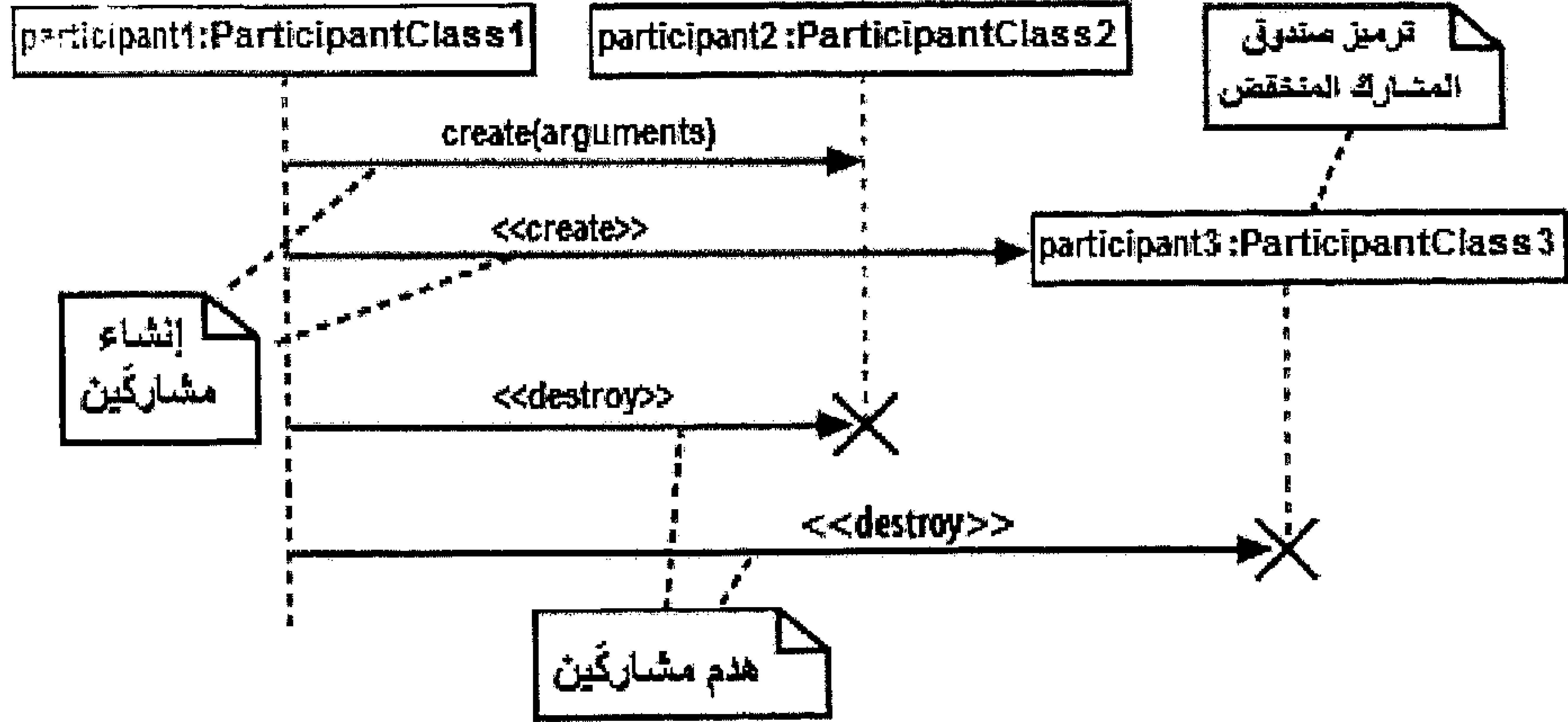
٧-٦-٤ رسائل إنشاء المشارك وتدميره

Participant Creation and Destruction Messages

لا يعيش المشاركون بالضرورة كامل مدة التفاعل في مخطط التتابع. ويمكن أن يتم إنشاء المشاركين و تدميرهم وفقاً للرسائل الممررة لهم، كما هو معروض في الشكل رقم (٧-١١).

ولعرض إنشاء مشارك ما، يمكن ببساطة إما تمرير الرسالة (..)create إلى خط حياته، أو استعمال ترميز صندوق المشارك المنخفض عندما يكون واضحاً تماماً عدم وجود المشارك قبل القيام باستدعاء رسالة الإنشاء. ويتم عرض تدمير المشارك بواسطة إنهاء خط حياته باستعمال شعار التدمير X.

ويتم إنشاء المشاركين برمجياً بلغة جافا و C# باستعمال الكلمة المفتاح new، كما هو معروض في المثال رقم (٧-٣).



شكل رقم (٧-١١) تم إنشاء المشارك participant2 و المشارك participant3 ضمن مسار مخطط التتابع المعروض.

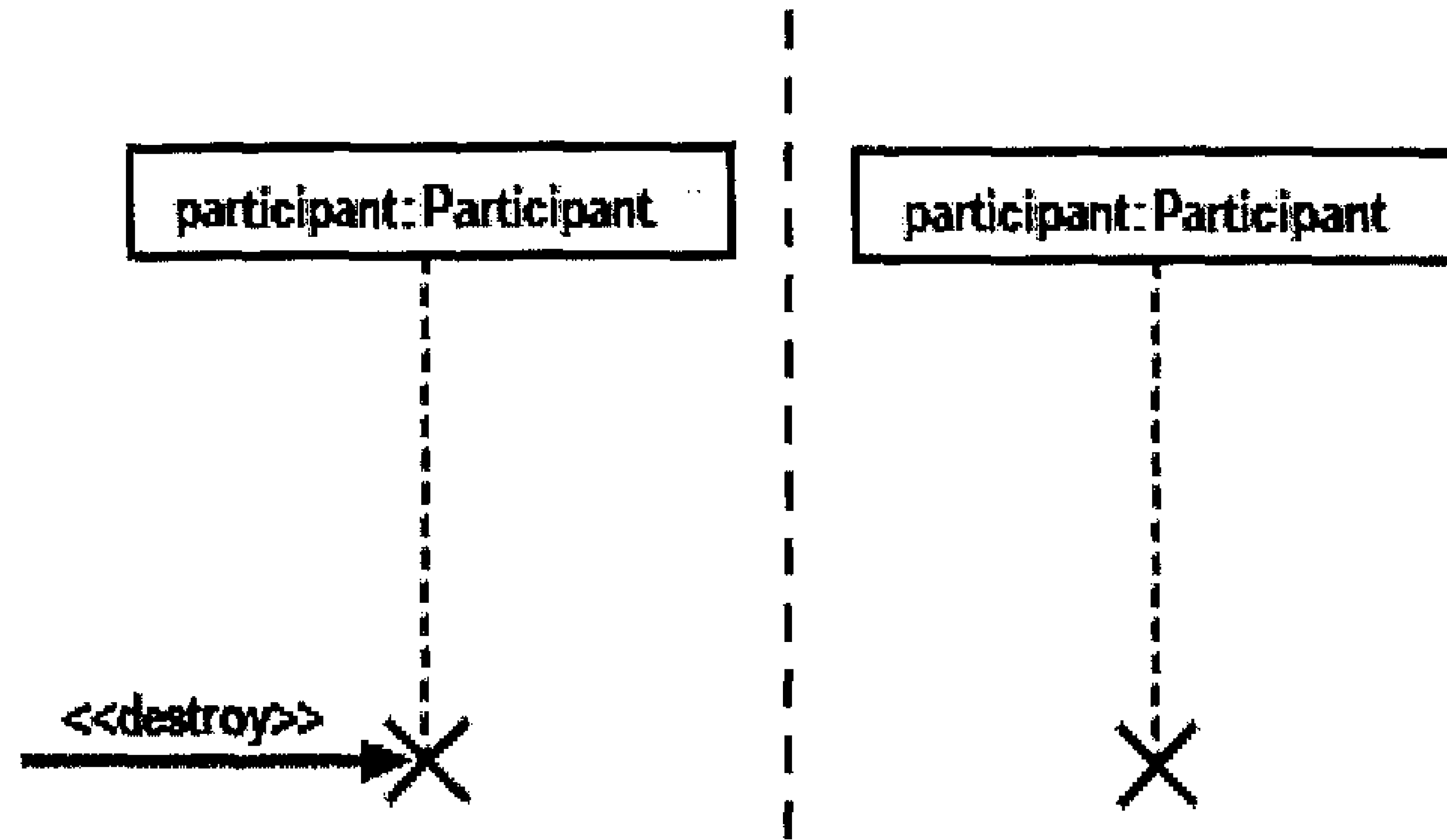
مثال رقم (٧-٣) يقوم الصنف MessageCaller بإنشاء كائن جديد من النوع MessageReceiver باستعمال الكلمة المفتاح new.

```
public class MessageReceiver {
  // MessageReceiver خصائص و طرق الصنف
}
public class MessageCaller {
  // يصرح عن الخصائص و الطرق الأخرى للصنف هنا
  public void doSomething( ) {
    // MessageReceiver يتم إنشاء كائن
    MessageReceiver messageReceiver = new MessageReceiver( );
  }
}
```

مع بعض لغات البرمجة، مثل لغة جافا، ليس لدينا طريقة لتدمير الكائن بشكل صريح، لذلك لا يوجد معنى لإظهار رسالة التدمير في

مخطط التتابع. ويجسد المثال رقم (٧-٣) هذه الحالة، حين ينتهي تنفيذ الطريقة ()doSomething سيتم تحديد الكائن messageReceiver لتدميره لاحقا، ولا يجب تمرير رسائل إضافية إلى messageReceiver لجعله يدمر نفسه بسبب إدارة ذلك ضمنياً من قبل مجمع القمامة garbage collector بلغة جافا.

في هذه الحالات، عند وجود عامل آخر يتعلق بأمر التدمير، مثل مجمع القمامة، يمكن إما ترك الكائن نشطاً لكن غير مستعمل، أو الإشارة إلى أنه لم يعد ضرورياً باستعمال رمز التدمير X من دون إرفاقه بطريقة التدمير ()destroy، كما هو معروض في الشكل رقم (٧-١٢).

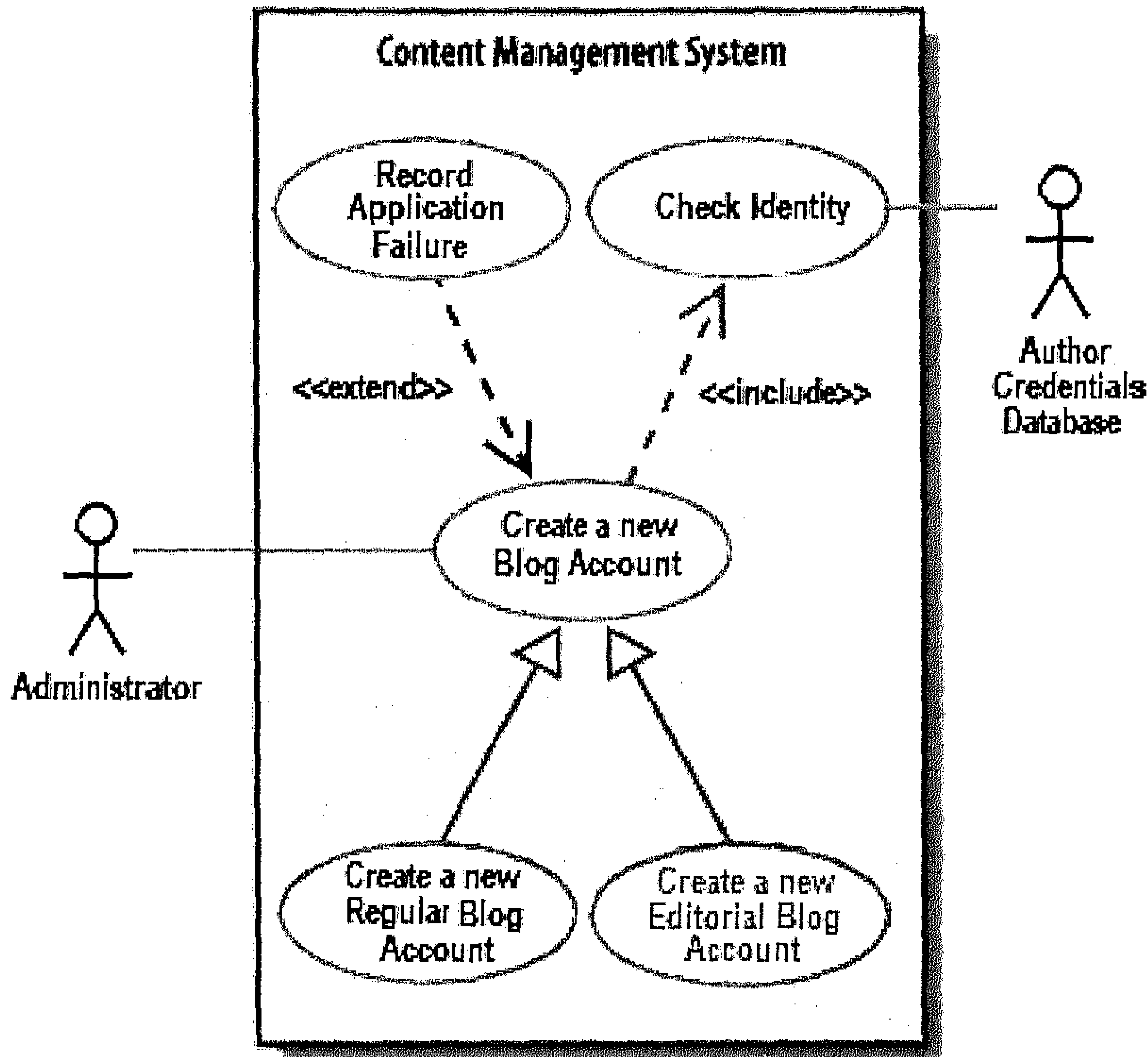


شكل رقم (٧-١٢) استعمال رسالة تدمير واضحة أو الإشارة إلى إهمال المشارك بمجرد استعمال رمز التدمير X.

٧-٧ بث الحياة في حالات الاستخدام مع مخطط التتابع
Bringing a Use Case to Life with a Sequence Diagram

حان الوقت لإلقاء نظرة أقرب على التتابع. و بشكل خاص، دعنا ننظر إلى مخطط التتابع الذي سيقوم بنمذجة التفاعلات الضروري حدوثها لجعل حالة الاستخدام "إنشاء حساب مدونة عادي جديد" تحدث.

يجب أن يبدو الشكل رقم (٧-١٣) مألوفاً لك؛ فهو مجرد تذكير سريع لحالة الاستخدام "إنشاء حساب مدونة عادي جديد" (انظر إلى الفصل الثاني).



شكل رقم (٧-١٣) مخطط حالة الاستخدام "إنشاء حساب مدونة عادي جديد".

باختصار، تشكل حالة الاستخدام "إنشاء حساب مدونة عادي جديد **Create a new Regular Blog Account**" حالة خاصة من حالة الاستخدام "إنشاء حساب مدونة جديد **Create a new Blog Account**". وتتضمن أيضاً كل الخطوات المزودة من قبل حالة الاستخدام "التحقق من الهوية **Check Identity**"، وقد تنفذ بشكل اختياري الخطوات المزودة من

قبل حالة الاستخدام "تسجيل فشل التطبيق Record Application Failure"، عند رفض طلب إنشاء حساب جديد. يعتبر الشكل رقم (٧-١٣) مخطط حالة استخدام نشط جداً، لذلك يمكنك الرجوع إلى الفصل الثاني لتتذكر ما يحصل في هذا المخطط.

دعم أسلوب خفض صندوق عنوان المشارك
Supporting the Dropped Title Box Technique
من المحزن عدم دعم العديد من أدوات لغة النمذجة الموحدة القياسية أسلوب خفض صندوق عنوان المشارك من أعلى المخطط، وذلك لإظهار إنشاء مشارك أو تدميره باستعمال الرمز X. على سبيل المثال، غالباً ما تجد أن الأداة المستعملة لا تسمح بوضع صندوق عنوان مشارك في أي مكان غير أعلى المخطط. في هذه الحالات، تكون الطريقة المثلى لإجراء ذلك بتبيان أن رسالة الإنشاء أو التدمير تستدعي الكائن المُنشأ، ويتم الاعتماد على قارئ المخطط لإدراك أننا نعني بذلك إنشاء مشارك (غالباً ما يفيد استعمال أيضاً ملاحظة لهذا الأمر). لسوء الحظ، لا تشكل هذه الطريقة أفضل استعمال للغة النمذجة الموحدة، لكن هذا كل ما يمكننا فعله مع الأداة.

٧-٧-١ مخطط تتابع عالي المستوى
A Top-Level Sequence Diagram

قبل التمكن من تحديد أنواع التفاعلات التي ستحدث عند تنفيذ حالة الاستخدام، نحتاج إلى توصيف مفصّل عمّا تعمله حالة الاستخدام. إذا قمت بإتمام توصيف حالة الاستخدام، يصبح عندك مرجع جيد عن هذه المعلومات المفصلة.

يعرض الجدول رقم (٧-٣) الخطوات التي تحدث خلال حالة الاستخدام "إنشاء حساب مدونة عادي جديد" طبقاً لتوصيفها المفصل.

في الحقيقة يظهر الجدول رقم (٧-٣) كل الخطوات المتعلقة بحالة الاستخدام "إنشاء حساب مدونة عادي جديد"، بما فيها الخطوات الموروثة

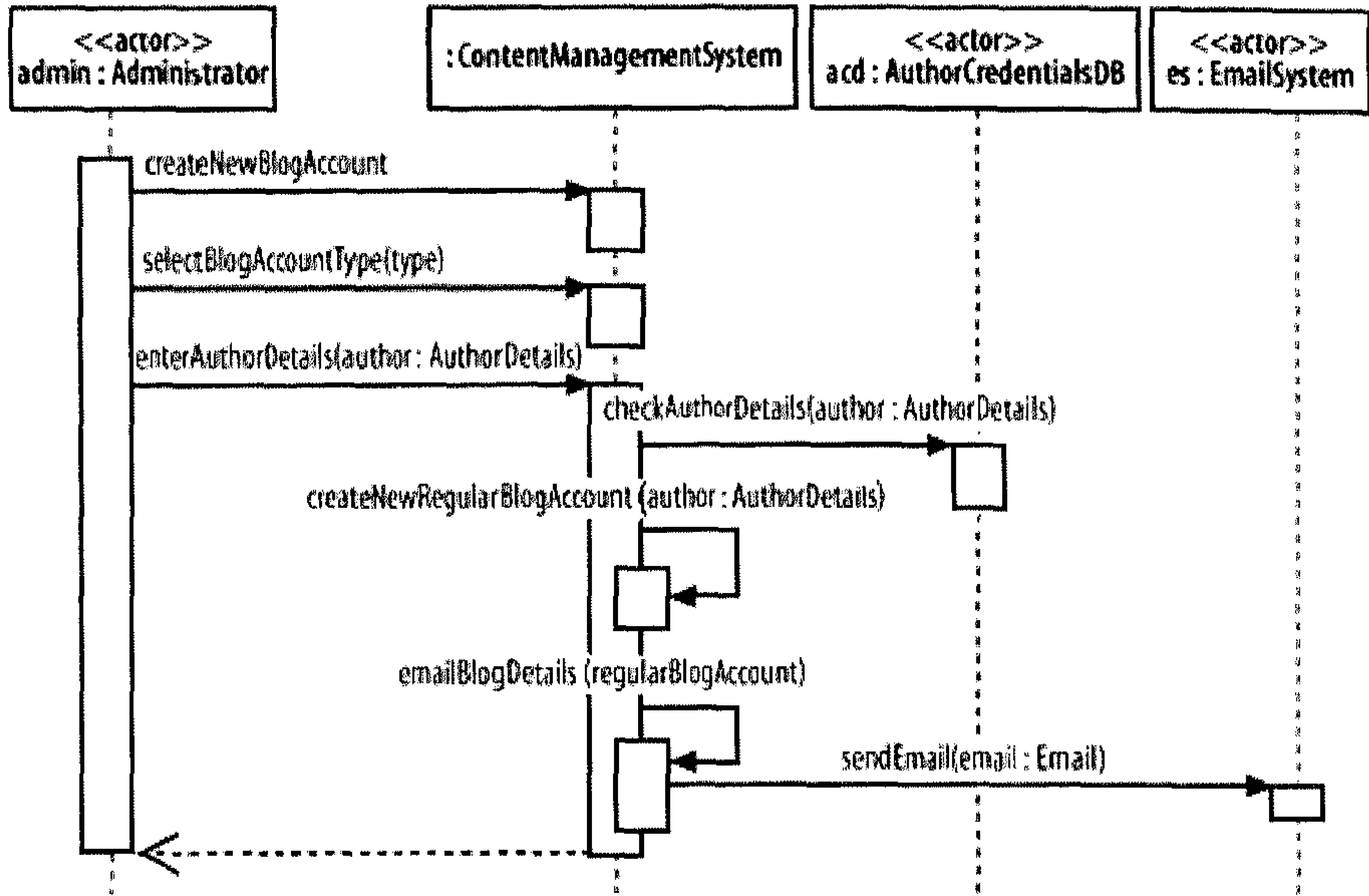
من حالة الاستخدام "إنشاء حساب مدونة جديد" أو المعاد استعمالها من حالة الاستخدام "التحقق من الهوية". ولقد تم إجراء ذلك لمجرد التمكن من رؤية كل خطوات التدفق الرئيسي في مكان واحد بشكل سهل.

جدول رقم (٧-٣) يجب أن تكون أغلب المعلومات التفصيلية الضرورية متوفرة سابقاً كتدفق رئيسي داخل توصيف حالة الاستخدام للبدء ببناء مخطط تتابع لها.

التدفق الرئيسي	الخطوة	العمل
	١	يطلب المدير من النظام إنشاء حساب مدونة جديد.
	٢	يختار المدير النوع حساب مدونة عادي.
	٣	يدخل المدير تفاصيل الكاتب.
	٤	يتم التحقق من تفاصيل الكاتب باستعمال قاعدة بيانات اعتماد الكتبة.
	٥	تم إنشاء حساب مدونة عادي جديد.
	٦	تم إرسال بريد إلكتروني للكاتب يضم ملخصاً عن تفاصيل حساب المدونة الجديد.

ربما تريد مجرد النظر في توصيفات حالات الاستخدام الثلاثة بشكل منفصل من دون الانزعاج بالتفكير بدمجهم فعلياً.

يعرض الجدول رقم (٧-٣) التدفق الرئيسي فقط - عبارة عن الخطوات التي ستحدث من دون الاهتمام أي توسيعات - ولكن تشكل هذه نقطة بداية كافية جداً لإنشاء مخطط تتابع عالي المستوى، كما هو معروض في الشكل رقم (٧-١٤).



شكل رقم (٧-١٤) يظهر مخطط التتابع المستخدمين المتفاعلين مع النظام، و يتم إظهار النظام ببساطة كجزء بسيط في التتابع.



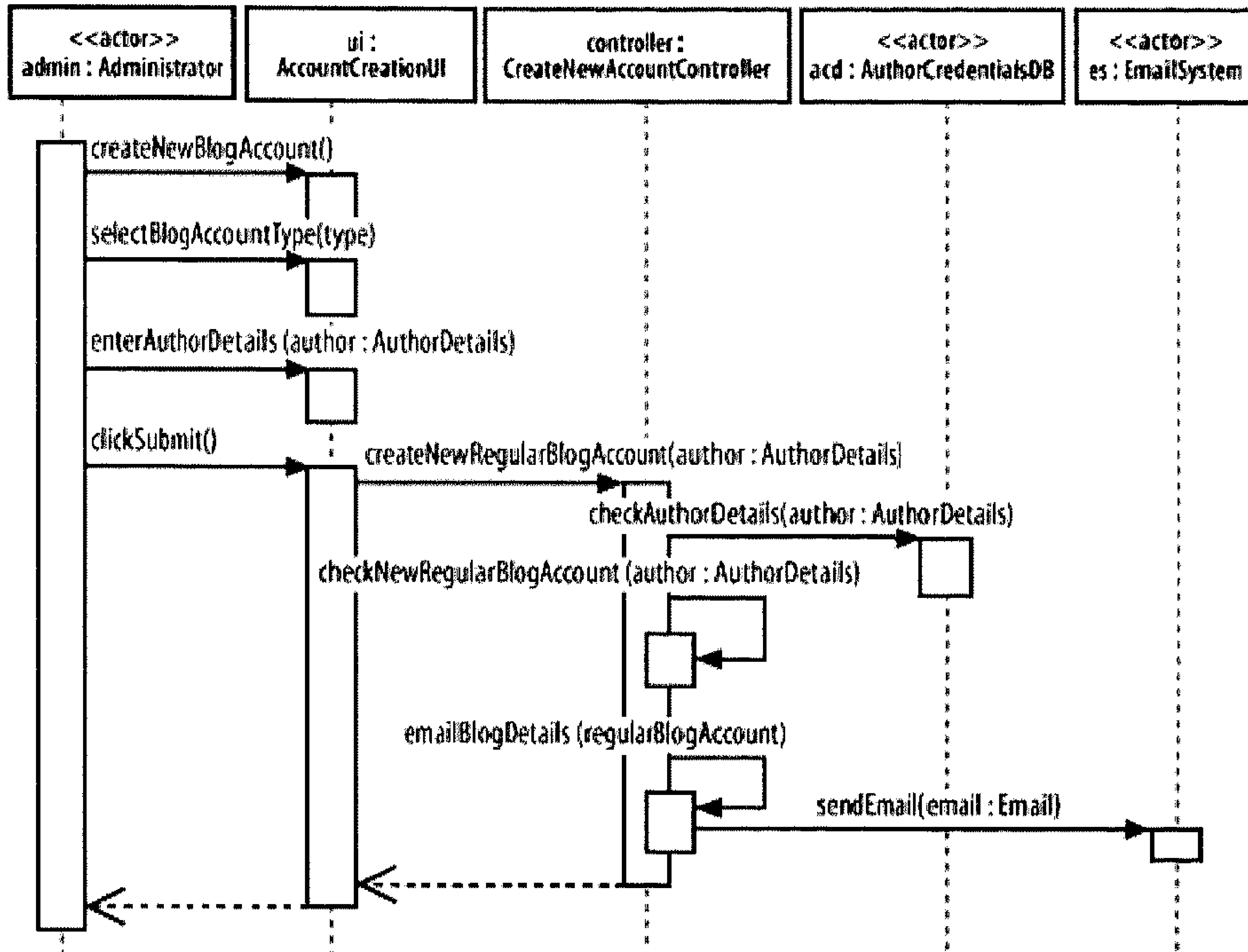
يركز الشكل رقم (٧- ١٤) على المشاركين و الرسائل المتعلقة بحالة الاستخدام. لقد تم نمذجة نفس حالة الاستخدام في الفصل الثالث كمخطط نشاط والذي يركز على العمليات ذات الصلة بدلاً من التركيز على المشاركين.

٧-٧-٢ تجزئة التفاعل إلى مشاركين منفصلين

Breaking an Interaction into Separate Participants

عند هذه النقطة، يعرض الشكل رقم (٧-١٤) التفاعلات التي يجب حدوثها بين المستخدمين الخارجيين والنظام فقط، لأنه عند هذا المستوى تم كتابة خطوات توصيف حالة الاستخدام. ولقد تم تمثيل النظام في مخطط التتابع كمشارك بسيط ContentManagementSystem؛ وعلى أية حال، إذا لم تكن تنوي إنجاز نظام إدارة المحتوى كشفرة كبيرة

وحيدة (و هي ليست فكرة جيدة عموما!)، فقد حان الوقت لتفكيك النظام ContentManagementSystem وإظهار الأجزاء التي بداخله، كما هو معروض في الشكل رقم (٧-١٥).



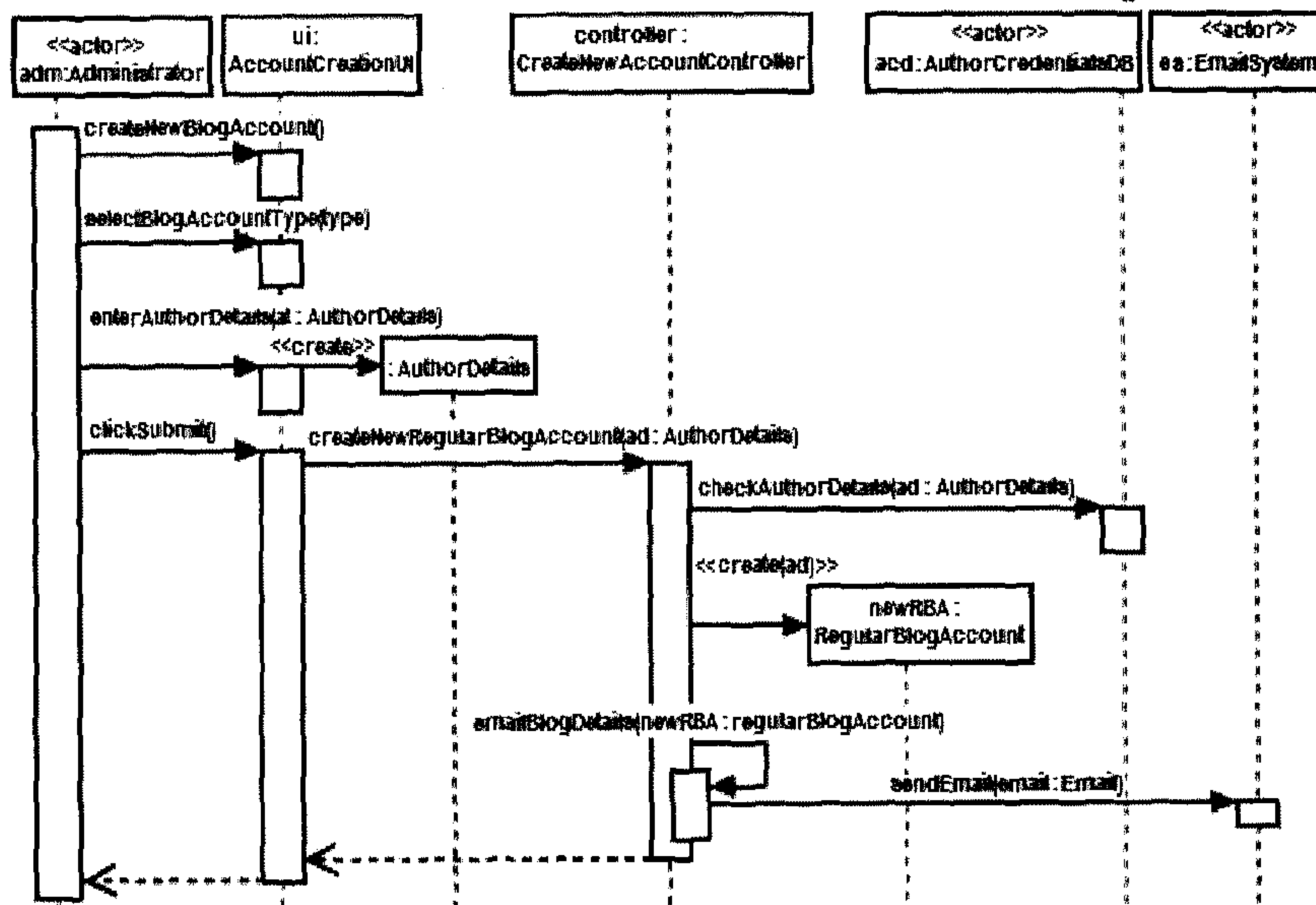
شكل رقم (٧-١٥) إضافة تفاصيل أكثر بخصوص الأجزاء داخل النظام.

يمكن أن تصبح مخططات التتابع كثيرة التعقيد، وذلك بإضافة مشاركين إضافيين وبعض التفاعلات الأكثر تفصيلاً. وفي الشكل رقم (٧-١٥)، تم تهذيب مخطط التتابع الأساسي بشكل أدى إلى حذف المشارك ContentManagementSystem وإضافة تفاصيل أكثر مكانه لعرض المشاركين الفعليين ذات الصلة.

ويستمر العمل على مخططات التتابع بشكل دائم طيلة حياة نموذج النظام، والحصول من البداية على المشاركين والتفاعلات الصحيحة في مخطط تتابع مفصل هو عمل شاق. ويشكل إبقاء مخططات التتابع محدثة أيضاً تحدياً عظيماً (انظر إلى "إدارة التفاعلات المعقدة مع أجزاء التتابع" لاحقاً في هذا الفصل)؛ لذلك، من المتوقع بذل بعض الوقت بالعمل على مخططات التتابع حتى الحصول على أشياء مناسبة.

٧-٧-٣ تطبيق إنشاء مشارك Applying Participant Creation

يوجد أمر حرج ناقص من مخطط التتابع المعروض في الشكل رقم (٧-١٥). ويعمل المخطط ضمن حالة الاستخدام "إنشاء حساب مدونة عادي جديد"، ولكن أين هو الإنشاء الفعلي لحساب المدونة؟ يضيف الشكل رقم (٧-١٦) الأجزاء الناقصة في النموذج لإظهار الإنشاء الفعلي لحساب المدونة العادي.



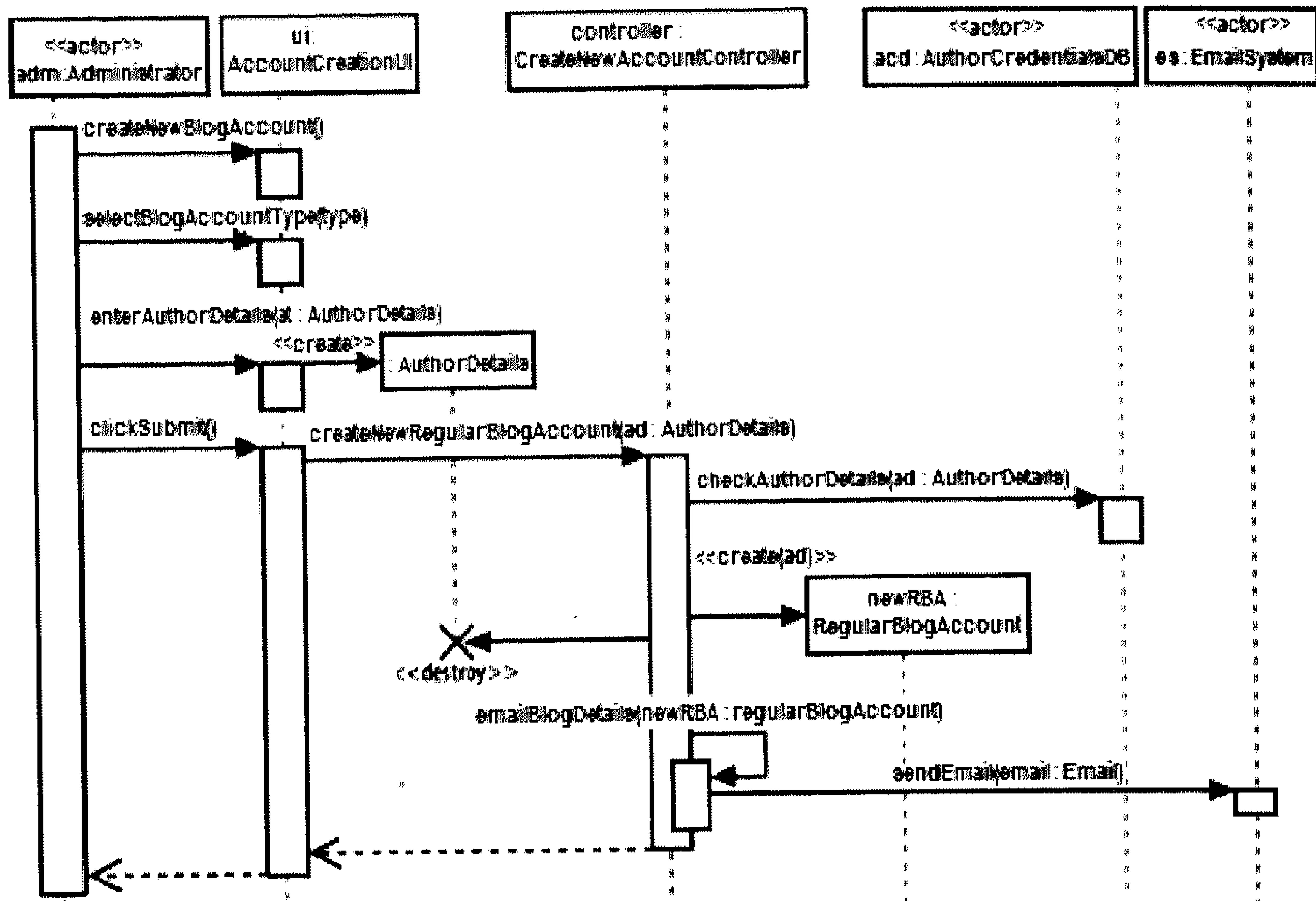
شكل رقم (٧-١٦) يعرض خطوط حياة المشاركين في مخطط التتابع.

تفيد خطوط حياة المشاركين كثيراً لإظهار موقع إنشاء المشارك. في الشكل رقم (٧-١٦)، لا يوجد المشارك AuthorDetails و لا المشارك RegularBlogAccount عند بداية مخطط التتابع، لكنه تم إنشاؤهما أثناء تنفيذ المخطط.

لقد تم إنشاء المشارك AuthorDetails والمشارك newAccount: RegularBlogAccount بواسطة رسالتي إنشاء create. وترتبط كل رسالة إنشاء بصندوق عنوان المشارك المُنشأ مباشرة، ويتم تمرير أية معلومات ضرورية عند إنشاء المشارك الجديد. ويتم خفض صندوق عنوان المشارك إلى النقطة التي يتم عندها استدعاء رسالة الإنشاء create، ويمكن أن يظهر المخطط بوضوح النقطة التي يبدأ عندها خط حياة المشارك.

٧-٧-٤ تطبيق تدمير المشارك Applying Participant Deletion

دعنا نقول أن المشارك authorDetails:AuthorDetails لم يعد متطلباً حالما يتم إنشاء المشارك newAccount:RegularBlogAccount. لإظهار تدمير المشارك authorDetails:AuthorDetails عند هذه النقطة، ويمكن استعمال رسالة تدمير واضحة متصلة برمز التدمير X، كما هو معروض في الشكل رقم (٧-١٧).

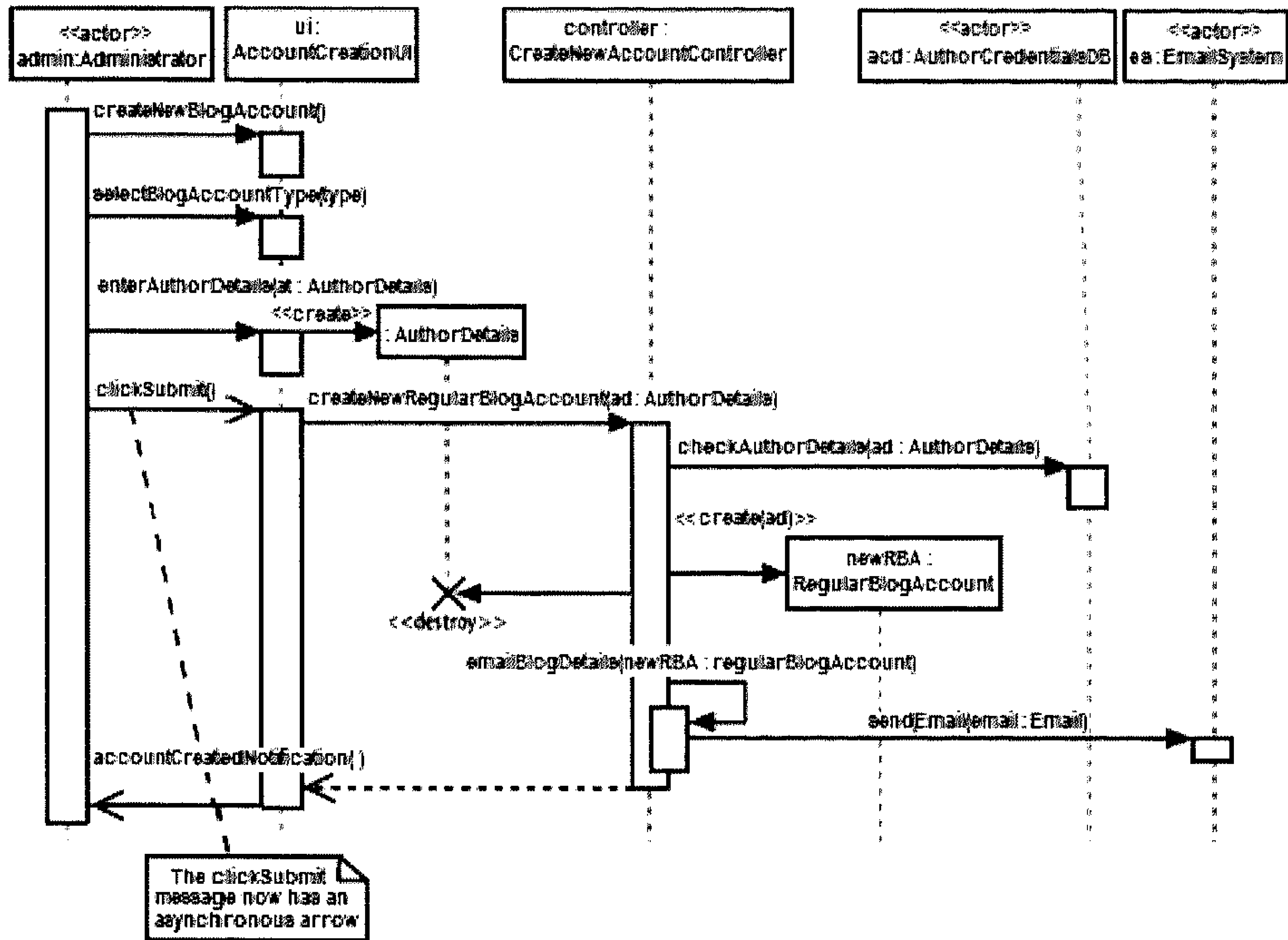


شكل رقم (٧-١٧) يظهر إهمال مشارك باستعمال رمز التدمير X.

٧-٧-٥ تطبيق الرسائل غير المتزامنة

Applying Asynchronous Messages

حتى الآن، كانت كل الرسائل التي في مثالنا عن مخطط التتابع متزامنة؛ أي يتم تنفيذها الواحدة بعد الأخرى بالترتيب، ولا يحدث أي شيء بشكل متنافس (بنفس الوقت). على أية حال، يوجد على الأقل رسالة واحدة في مثال التتابع كمرشح جيد لأن تكون رسالة غير متزامنة، كما هو معروض في الشكل رقم (٧-١٨).



شكل رقم (٧-١٨) ستنتج الرسالة clickSubmit() بعض السلوك غير العادي عندما يقوم المدير بإنشاء حساب جديد.

في الشكل رقم (٧-١٨)، عندما ينقر المدير Administrator على الزر أرسل submit سيتجمد النظام، و لن يسمح لك القيام بأي شيء حتى يتم إنشاء حساب مدونة جديد. ومن المفيد إظهار أن واجهة المستخدم تسمح للمدير Administrator بالاستمرار بالمهام الأخرى أثناء قيام نظام إدارة المحتوى بإنشاء الحساب الجديد. وما نحتاجه هنا هو تحويل الرسالة clickSubmit() إلى رسالة غير متزامنة.

ويعني تحويل الرسالة clickSubmit() من رسالة متزامنة إلى رسالة غير متزامنة أن مخطط التتابع سيظهر أنه لن يتم إقفال واجهة المستخدم عند إرسال معلومات حساب المدونة العادي الجديد، ولا حتى انتظار إتمام

إنشاء الحساب الجديد. و بدلاً من ذلك، ستسمح واجهة المستخدم للمدير Administrator بالاستمرار بالعمل على النظام.

وحتى يستلم المدير Administrator تغذية رجعية feedback تخبر عن مدى نجاح إنشاء حساب المدونة الجديد من عدمه، يجب استبدال سهم الرجوع البسيط بالرسالة غير المتزامنة الجديدة accountCreationNotification() لأن الرسائل غير المتزامنة لا ترجع قيماً.

٧-٨ إدارة التفاعلات المعقدة باستخدام أقسام التتابع

Managing Complex Interactions with Sequence Fragments

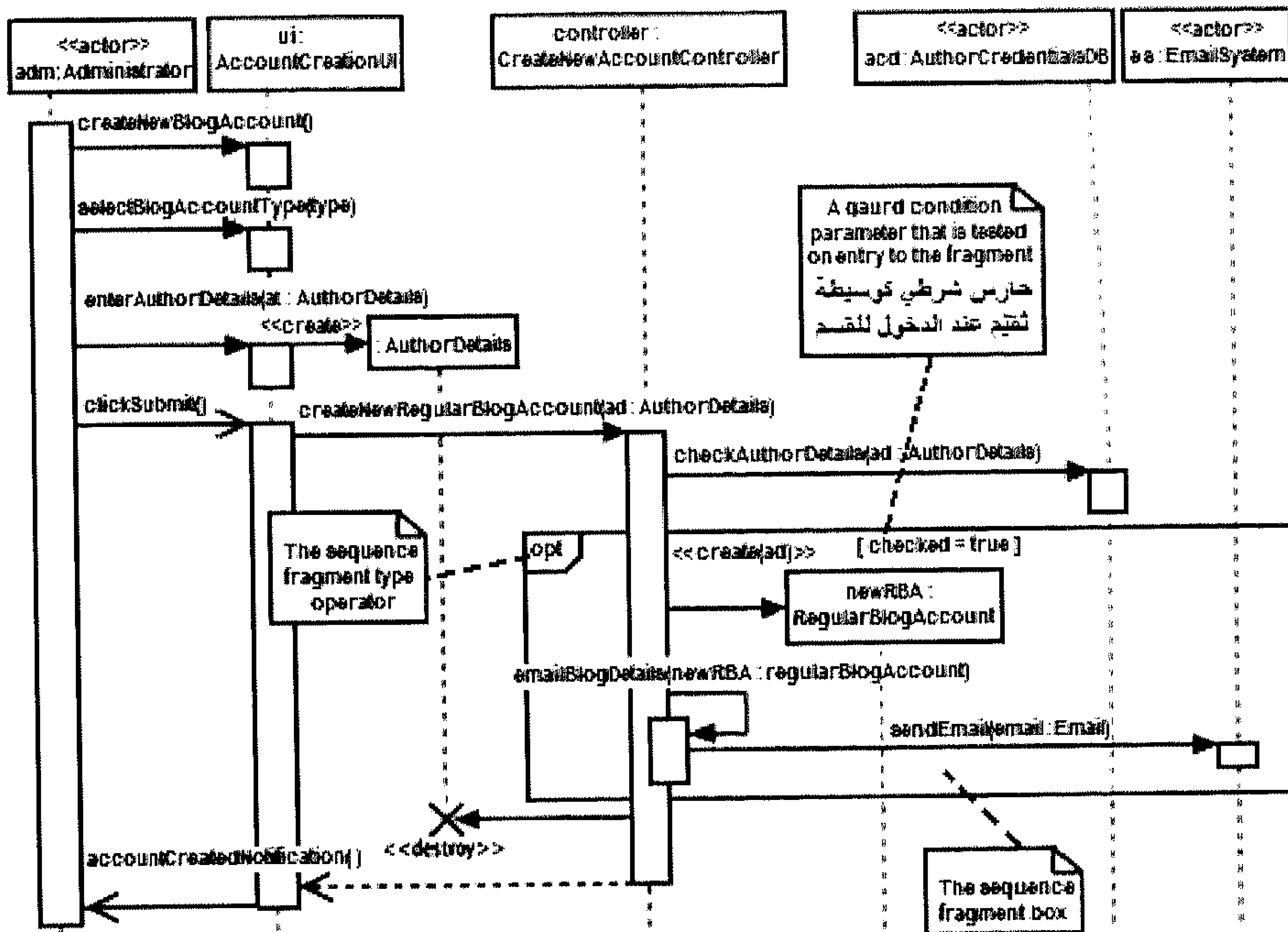
إن أغلب ما رأيته في هذا الفصل كان مألوفاً جداً لأي شخص قام باستعمال مخططات التتابع في UML 1.x، ولكن حان الوقت لرؤية شيء مختلف تماماً.

في الفترة السابقة لـ UML 2.0، كانت مخططات التتابع تصبح بشكل سريع ضخمة وغير منظمة، وتحتوي أكثر مما ينبغي من التفاصيل ليسهل فهمها وصيانتها. ولم تكن هناك طرق قياسية مبنية داخلياً لإظهار تدفقات التكرار والتفرع، لذلك كان عليك "تنمية حلولك الخاصة" بهذا الشأن. ويميل هذا الأمر إلى المساهمة في زيادة حجم مخططات التتابع وتعقيدها وبدلاً من المساعدة في إدارتها.

كان هناك حاجة إلى شيء جديد، لمساعدة القائمين بالنمذجة بالتعامل مع التفاصيل التي يجب أن يأسرها مخطط التتابع، مما يسمح لهم بإنشاء مخططات تتابع منظمة ومهيكلة تعرض تفاعلات معقدة، مثل تدفقات التكرار والتفرع. وقد أتى مصممو UML 2.0 بأقسام التتابع **sequence fragment** للإجابة عن هذه الأمور.

ويتم تمثيل قسم التتابع على شكل صندوق يحيط بقسم من التفاعلات التي داخل مخطط التتابع، كما هو معروض في الشكل رقم (٧-١٩).

ويمتد صندوق قسم التتابع على منطقة من مخطط التتابع، حيث تأخذ تفاعلات قسم التتابع مكانها داخل هذه المنطقة. ويمكن لصندوق قسم التتابع أن يحتوي على أي عدد من التفاعلات، وأن يحتوي أيضاً على أقسام تتابع داخلية مع التفاعلات المعقدة الكبيرة. وتحتوي الزاوية العليا على يسار صندوق قسم التتابع على عامل operator. ويشير عامل القسم **fragment operator** إلى نوع قسم التتابع.



شكل رقم (٧-١٩) تم تحديد قسم التتابع كجزء من مخطط تتابع أكبر، و تم استعمال الملاحظات لتحديد صندوق قسم التتابع و أي بارامترات وعامل القسم.

في الشكل رقم (٧-١٩)، يلاحظ أنَّ عامل القسم هو العامل opt، مما يعني أنه عندنا قسم تتابع اختياري optional. أي سيتم تنفيذ كل التفاعلات داخل صندوق قسم التتابع الاختياري وفقاً لنتيجة بارامتر قسم التتابع الذي يؤدي دور الحارس الشرطي.

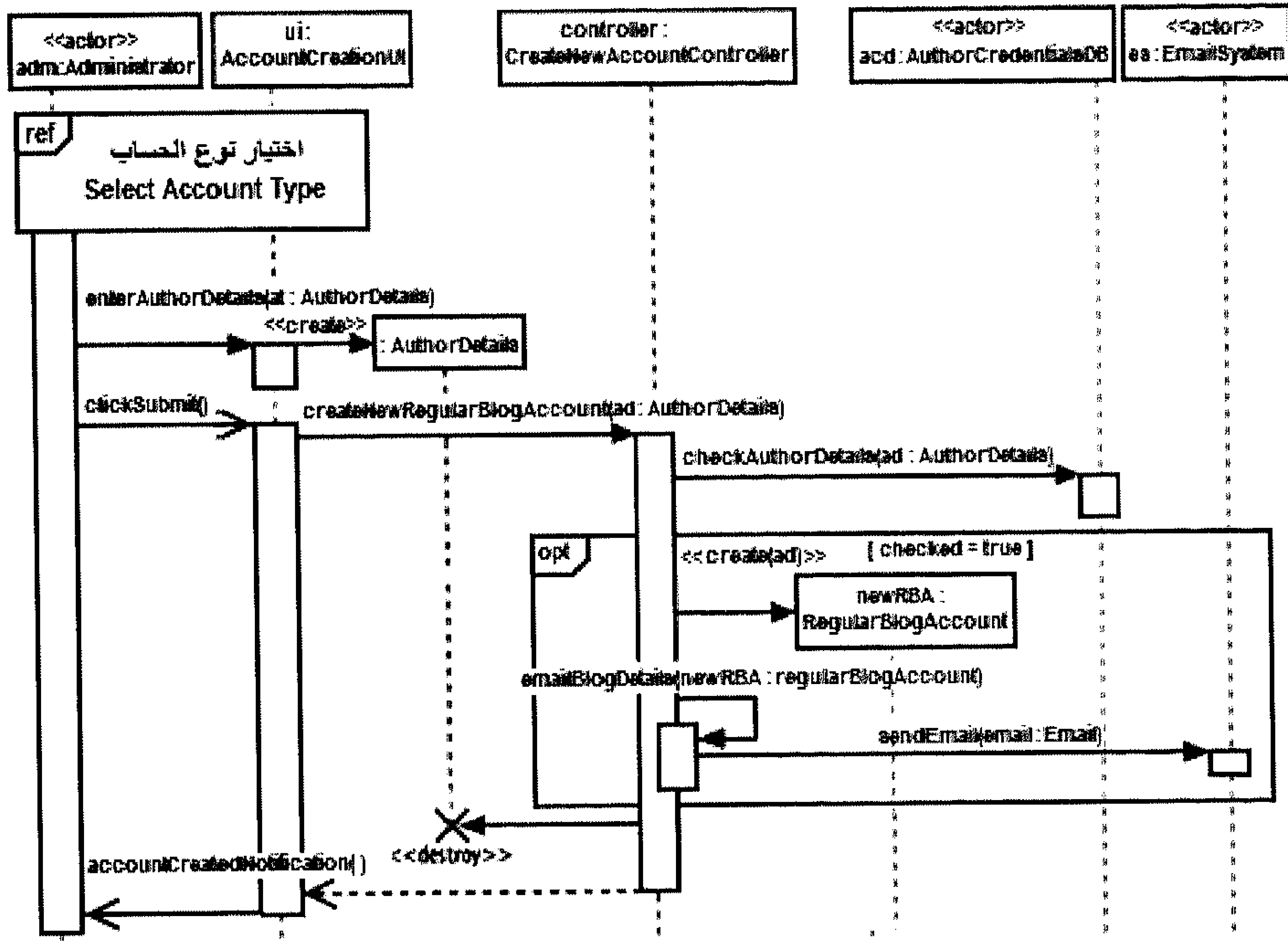
ولا تحتاج بعض أنواع أقسام التتابع إلى بارامترات إضافية كجزء من توصيفاتها، مثل النوع المرجعي ref لقسم التتابع الذي سيناقش في الجزء القادم، ولكن يحتاج النوع opt لقسم التتابع إلى بارامتر حارس شرطي لأخذ القرار: إذا ما كان عليه تنفيذ تفاعلاته أم لا. في حالة نوع قسم التتابع opt، يتم تنفيذ التفاعلات التي في قسم التتابع عندما تكون قيمة الحارس الشرطي المناظر صحيحة true.

٧-٨-١ استعمال قسم التتابع: قسم التتابع المرجعي

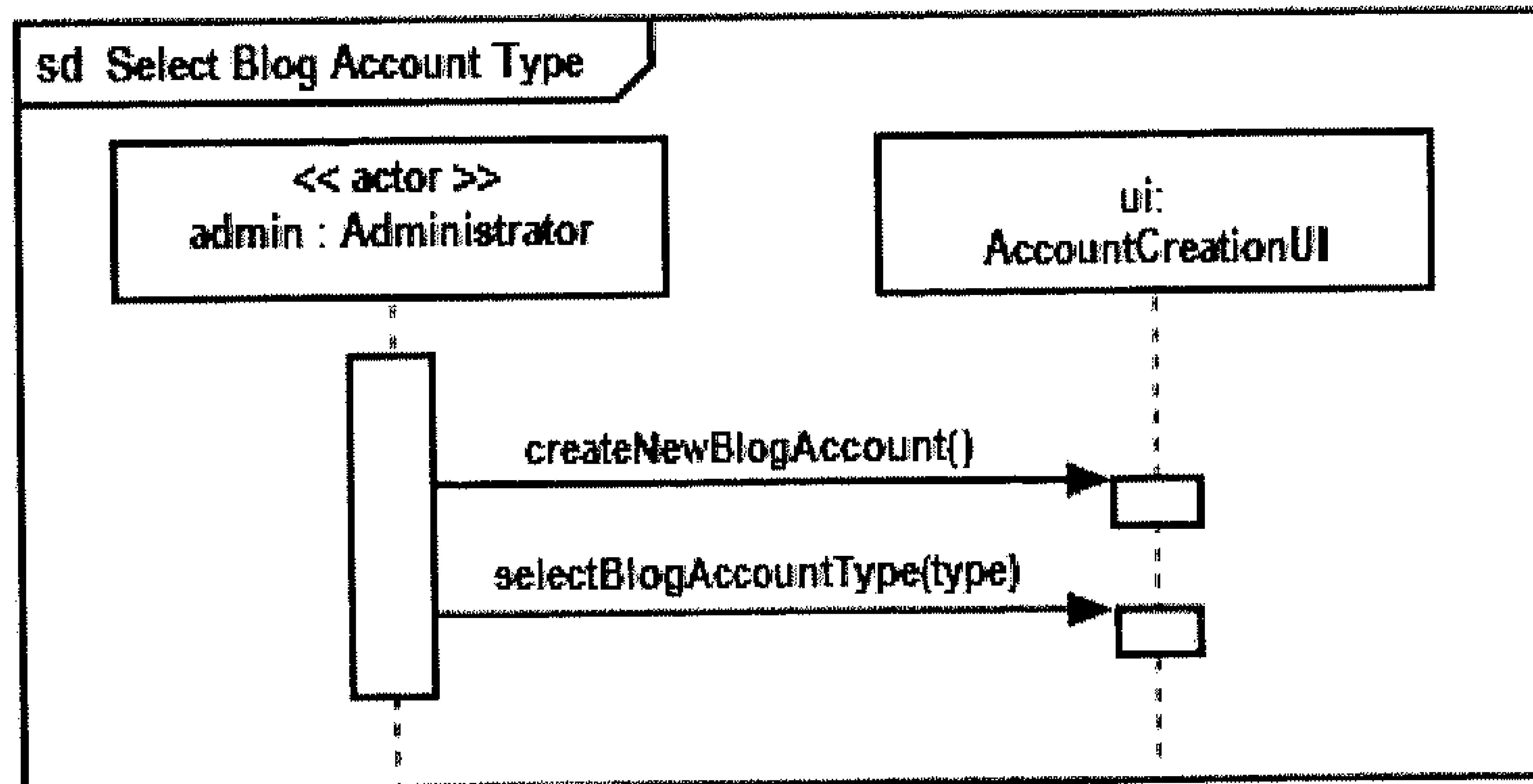
Using a Sequence Fragment: The ref Fragment-ref

يخفف النوع المرجعي ref لقسم التتابع بعضاً من مشاكل الصيانة الناتجة عن مخططات التتابع الضخمة، والتي يتم عادة إنشاؤها من أجل الأنظمة المعقدة. وفي الشكل رقم (٧-٢٠)، يمثل قسم التتابع المرجعي ref جزءاً من مخطط تتابع كبير.

لقد تم احتواء التفاعلات التي من خلالها يختار المستخدم مدير Administrator نوع حساب المدونة داخل قسم تتابع مرجعي. ويعرض الشكل رقم (٧-٢١) كيف يمكن تحديد قسم التتابع المرجعي على مخطط تتابع منفصل.



شكل رقم (٧-٢٠) أسر التفاعلات المستعملة في اختيار نوع حساب مدونة داخل قسم تتابع مرجعي.



شكل رقم (٧-٢١) مخطط تتابع مرجعي يحتوي على تفاعلات اختيار حساب مدونة جديد.

بالتوازي مع إدارة حجم مخططات التتابع الكبيرة، يقدم قسم التتابع المرجعي ref أيضاً فرصة لإعادة استعمال مجموعة تفاعلات مشتركة، وبهذه الوسيلة يعاد استعمال التفاعلات في عدة أماكن.



يعمل النوع المرجعي ref لقسم التتابع بشكل مشابه جداً لعلاقة التضمين <<include>> مع حالات الاستخدام. انظر إلى الفصل الثاني للمزيد عن علاقة التضمين مع حالات الاستخدام.

٧-٨-٢ ملخص مختصر عن أنواع أقسام التتابع مع لغة النمذجة الموحدة ٢,٠

A Brief Overview of UML 2.0's Fragment Types

تحتوي UML 2.0 على مجموعة واسعة من أنواع أقسام التتابع يمكن تطبيقها على مخططات التتابع لجعلها أكثر تعبيراً، كما هو معروض بالجدول رقم (٧-٤).

جدول رقم (٧-٤) أنواع أقسام التتابع وفائدتها عند إنشاء مخططات التتابع.

نوع قسم التتابع	البارامترات	لماذا هو مفيد؟
مرجع: ref	لا شيء	يمثل تفاعلاً معرّفاً بمكان آخر في النموذج. ويساعد في إدارة مخطط واسع من خلال تقسيمه، وربما بإعادة استعماله لمجموعة من التفاعلات. يشبه إعادة استعمال النمذجة عند تطبيق علاقة التضمين <<include>> مع حالات الاستخدام
تأكيد: assert	لا شيء	يفرض وجوب حصول التفاعلات داخل صندوق قسم التتابع بالضبط كما تم تحديدها؛ وإلا يعلن عن القسم أنه غير سليم و يتم رفع استثناء بذلك. تعمل بطريقة مماثلة للتعليمة assert بلغة جافا. وتفيد في تحديد وجوب حدوث كل خطوة في التفاعل بنجاح، وبمعنى آخر، عند نمذجة معاملة تجارية.

نوع قسم التتابع	البارامترات	لماذا هو مفيد؟
تكرار: loop	المرات الأدنى المرات الأقصى [شرط_حارس]	يدور على التفاعلات داخل قسم التتابع بعدد محدد من المرات، حتى تصبح قيمة شرط الحارس خطأ. وهو شبيه جداً بالتكرار (..)for بلغة جافا و #C. يفيد في تنفيذ مجموعة تفاعلات عدد محدد من المرات.
خروج: break	لا شيء	إذا حدثت التفاعلات داخل الجزء التابع له، يجب بالتالي الخروج من أي تفاعل يحيط بجزء التتابع الذي غالباً ما يكون جزء تتابع تكرار. تشبه تعليمة الخروج break بلغة جافا و #C.
تفرع متعدد: alt	[شرط_حارس١] [شرط_حارس٢] [غير ذلك]	يتعلق بأي حارس يقيّم أولاً على القيمة صح، حيث يتم تنفيذ المجموعة الفرعية من التعليمات المناظرة له. إنه يساعد في تحديد تنفيذ مجموعة تعليمات فقط وفقاً لبعض الشروط. وهو مشابه للتعليمة الشرطية البرمجية if-then- else.
اختيار: opt	[شرط_حارس]	تنفذ التفاعلات داخل قسم التتابع فقط إذا كانت قيمة شرط الحارس صح، تشبه تعليمة if بسيطة من دون else مناظرة لها. تفيد خاصة في إظهار الخطوات التي يتم إعادة استعمالها من قبل حالات استخدام أخرى بمخططات التتابع، حيث تكون علاقة حالات الاستخدام هنا هي علاقة التوسيع <<extend>>.
استبعاد: neg	لا شيء	يحدد أنه لن تنفذ التفاعلات داخل قسم التتابع مطلقاً. يساعد في تحديد أن مجموعة من التفاعلات لن تنفذ حتى تكون متأكداً من إمكانية حذفها. يفيد كثيراً عند استعمال أداة للغة النمذجة الموحدة قابلة للتنفيذ، حيث يكون مخطط التتابع ينفذ حالياً. يفيد أيضاً في إظهار عدم إمكانية إنجاز شيء ما، بمعنى آخر، عندما تريد إظهار عدم استطاعة مشارك ما استدعاء طريقة القراءة ()read على مقبس socket بعد غلقه باستدعاء طريقة الإغلاق ()close. يشبه استدعاء بعض الطرق بالبرمجة داخل تعليقات لاستبعادها من التنفيذ.
توازي: par	لا شيء	يحدد أنه يمكن تنفيذ التفاعلات التي بداخل قسم التتابع بشكل متوازٍ. يشبه القول أنه ليس هناك حاجة لأي إقفال لمسلك آمن متطلب داخل مجموعة من التفاعلات.
منطقة: region	لا شيء	يُقال للتفاعلات داخل هذا النوع من قسم التتابع أنها جزء من منطقة حرجة. المنطقة الحرجة هي المنطقة التي يتم عادة فيها تحديث مشترك مُشارك. إذا تم دمجها مع التفاعلات المتوازية المحددة باستعمال نوع قسم التتابع par، يمكن نمذجة أين لا تكون التفاعلات مسلكاً أو عملية - آمنة (قسم التتابع par) وأين يكون متطلباً الإقفال لمنع التفاعلات المتوازية من التداخل (قسم التتابع region). تشبه المقاطع المتزامنة وأقفال الكائنات بلغة جافا.

تجعل أقسام التتابع عملية إنشاء مخططات التتابع الدقيقة وصيانتها أكثر سهولة. على أية حال، من المفيد تذكّر أنه من دون أقسام التتابع يكون المخطط قطعة واحدة؛ ويمكن خلط وتنسيق أي عدد من أقسام التتابع لنمذجة التفاعلات على مخطط التتابع بشكل دقيق. يجب الحذر من أن تصبح المخططات ضخمة وغير عملية حتى عند استعمال أقسام التتابع، لأننا قد نكون ببساطة نحاول نمذجة أمور كثيرة في تتابع واحد.

لقد قدمنا ملخصاً موجزاً عن أقسام التتابع في مخططات التتابع. تشكل أنواع أو أقسام التتابع بحد ذاتها موضوعاً كبيراً وتخرج قليلاً عن نطاق هذا الكتاب. ومن أجل نظرة أشمل عن أنواع أقسام التتابع المختلفة، انظر إلى الكتاب UML 2.0 in a Nutshell (O'Reilly).

٧-٩ ما هي الخطوة التالية؟

تتعلق مخططات التتابع إلى حد كبير بمخططات الاتصال لدرجة أن عدداً من المنمذجين لا يعرفون عادة متى عليهم استعمال مخططات التتابع مقابل مخططات الاتصال. ويقوم الفصل الثامن بتقديم وصف لمخططات الاتصال ويختتم بمقارنة بين هذين النوعين من المخططات من خلال إعطاء بعض النصائح حول: متى يستعمل كل منها؟

إن مخططات التتابع و الاتصال هي مخططات تفاعل؛ ومخططات التوقيت هي أيضاً نوع آخر من مخططات التفاعل. وتختص مخططات التوقيت بعرض القيود الزمنية المرتبطة بالتفاعلات، وتكون مفيدة بشكل خاص مع الأنظمة الفورية real-time. وستتم تغطية مخططات التوقيت في الفصل التاسع.

إذا أصبحت مخططات التتابع غير منظمة و مزدحمة بكثير من الرسائل، ارجع إلى الوراء وانظر إلى مخططات التفاعل على مستوى أعلى مع مخططات ملخص التفاعل interaction overview diagrams. تنمذج مخططات ملخص التفاعل التصوّر الكبير لوجهة النظر عن التفاعلات التي تحصل داخل النظام. وسيتم وصف مخططات ملخص التفاعل في الفصل العاشر.

التركيز على روابط التفاعل: مخططات الاتصال

FOCUSING ON INTERACTION LINKS: COMMUNICATION DIAGRAMS

الهدف الرئيسي من مخططات التتابع هو إظهار ترتيب الأحداث بين أجزاء النظام المشتركة بتفاعل محدد. وتضيف مخططات الاتصال تصوراً آخر للتفاعل بالتركيز على الروابط التي بين المشاركين.

وتعمل مخططات الاتصال بشكل جيد خصوصاً لإظهار الروابط التي تكون ضرورية بين المشاركين لتمرير رسائل التفاعل. ومن خلال نظرة سريعة على مخططات الاتصال، يمكن معرفة المشاركين الذين يجب ربطهم ليصبح بالإمكان حدوث تفاعل ما.

وتكون الروابط بين المشاركين في مخطط التتابع ضمنية بفعل الرسائل الممررة بينهم. وتوفر مخططات الاتصال وسيلة بديهية لإظهار الروابط بين المشاركين الضروريين للأحداث المؤلفة للتفاعل. في مخططات الاتصال، يعتبر ترتيب الأحداث المشتركة في التفاعل - تقريباً - جزءاً ثانوياً من المعلومات.

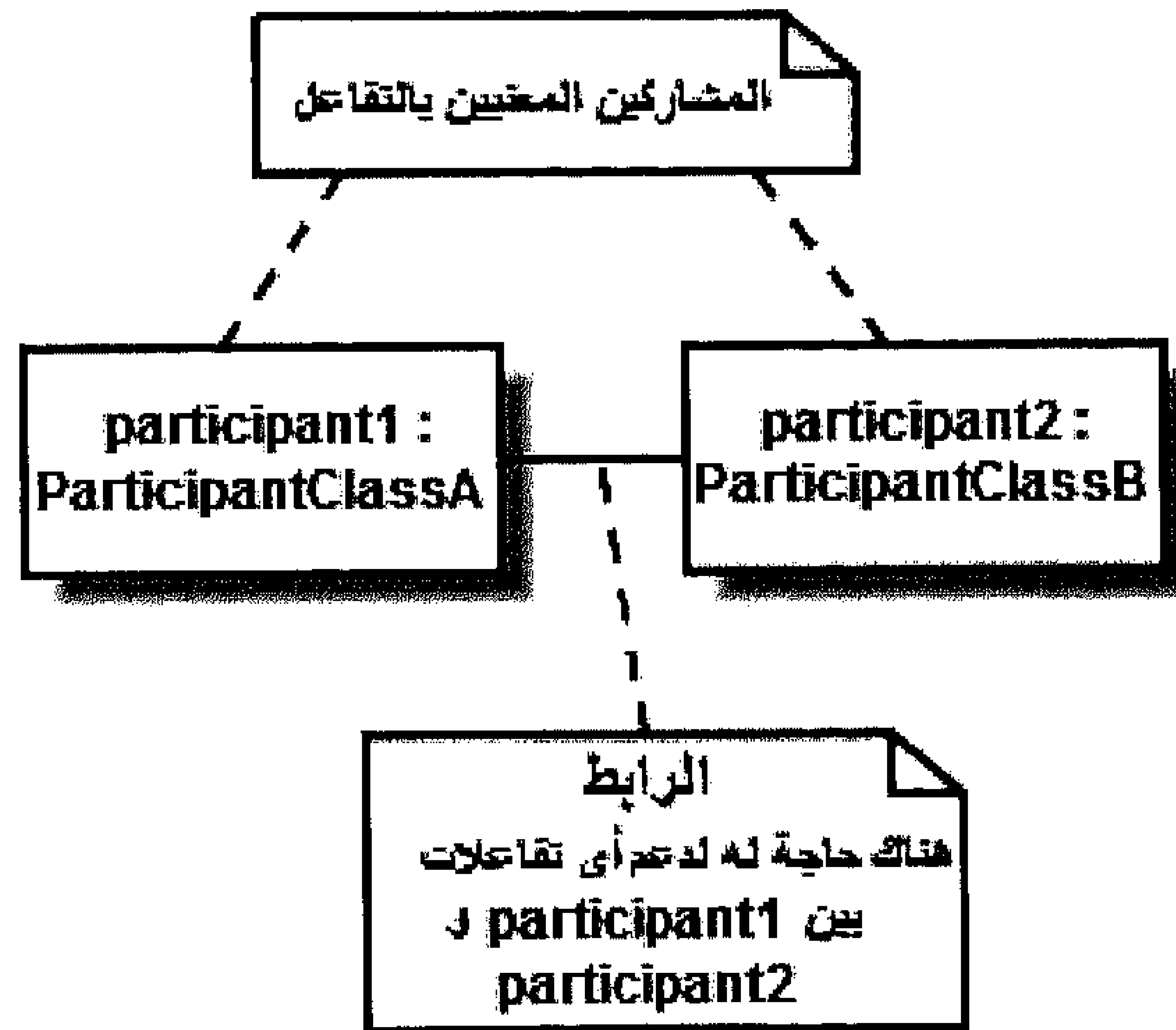


تتشابه مخططات التتابع و مخططات الاتصال كثيراً لدرجة تمكّن معظم أدوات لغة النمذجة الموحدة من التحويل الآلي من أحد هذين المخططَين إلى الآخر. وغالباً ما يكون الاختلاف بين هذين الأسلوبين عبارة عن تفضيل شخصي. وإذا نظرت إلى التفاعلات من منظور الروابط، ربما تكون مخططات الاتصال هي الأصلح لك؛ وعلى أية حال، إذا كنت تفضّل رؤية ترتيب التفاعلات بأوضح شكل ممكن، ربما يتحتم عليك التوجه أكثر نحو مخطط التتابع.

٨-١ المشاركين و الروابط و الرسائل

Participants, Links, and Messages

يتألف مخطط الاتصال من ثلاثة أشياء: المشاركين، وروابط الاتصال بينهم، والرسائل التي يمكن تمريرها على طول روابط الاتصال، كما هو معروض في الشكل رقم (٨-١).

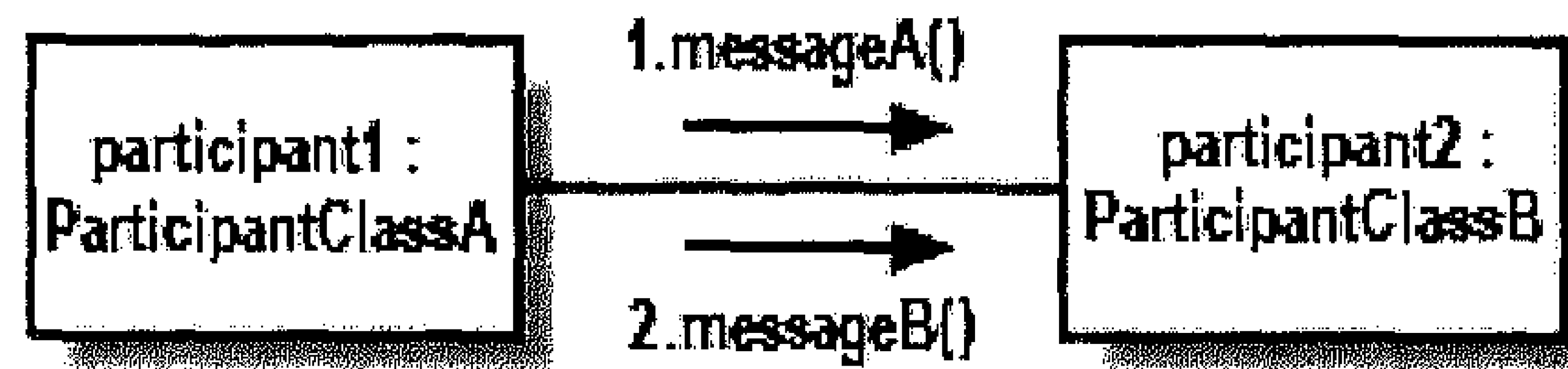


شكل رقم (٨-١) تتألف مخططات الاتصال من مشاركين و روابط بحيث أسهل بكثير من مخططات التتابع.

يتم تمثيل المشاركين في مخطط الاتصال بواسطة شكل المستطيل. ويوضع اسم المشارك وصنفه في وسط المستطيل. وتتكون صيغة اسم المشارك كالتالي: <name>: <class>، بشكل يشبه المشاركين في مخطط التتابع.

وتحتاج إلى تحديد اسم المشارك أو صنفه (أو كلاهما معاً). إذا لم يكن لديك معلومات عن اسم المشارك وصنفه معا، ويمكن بالتالي حذف إما صنفه أو اسمه. ويمكن أن يكون المشارك أحياناً مجهول الاسم anonymous.

ويتم إظهار رابط الاتصال بواسطة خط بسيط يربط بين مشاركين. الهدف من رابط الاتصال هو السماح بتمرير الرسائل بين مختلف المشاركين؛ من دون الرابط، ولا يمكن للمشاركين غير المرتبطين من التفاعل فيما بينهم. يظهر رابط الاتصال في الشكل رقم (٨-٢).



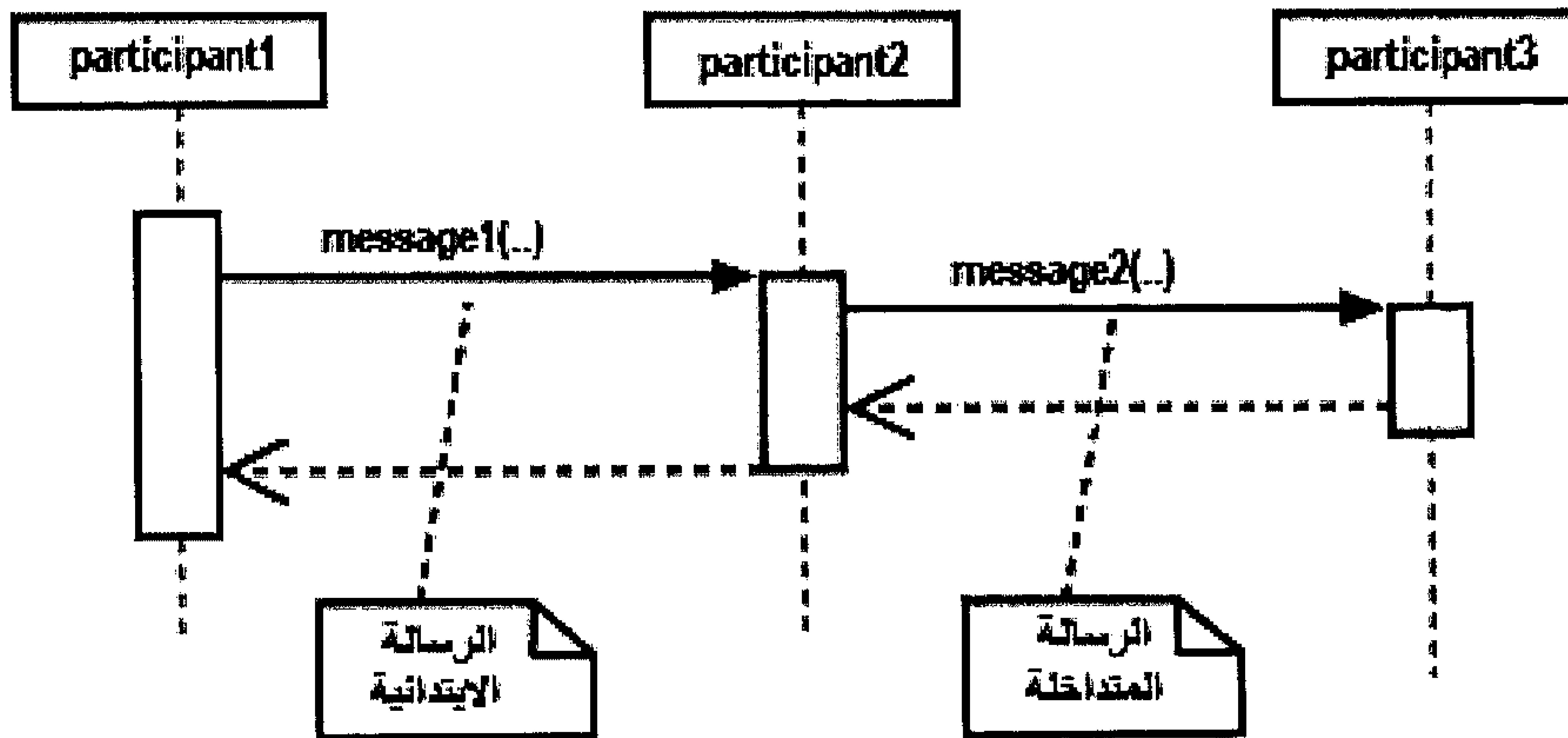
شكل رقم (٨-٢) تمرير رسالتين بمحاذاة الرابط بين المشارك **participant1** والمشارك **participant2**.

يتم إظهار الرسالة في مخطط الاتصال باستعمال سهم معبّأ الرأس من مرسل الرسالة إلى مستلمها. بأسلوب مشابه للرسائل في مخطط التتابع، يتألف توقيع الرسالة من اسمها و قائمة بارامترات. على أية حال، بخلاف مخططات التتابع، لا يكفي توقيع الرسالة وحده مع مخطط

الاتصال حيث نحتاج أيضاً إلى إظهار ترتيب استدعاء الرسائل أثناء التفاعل.

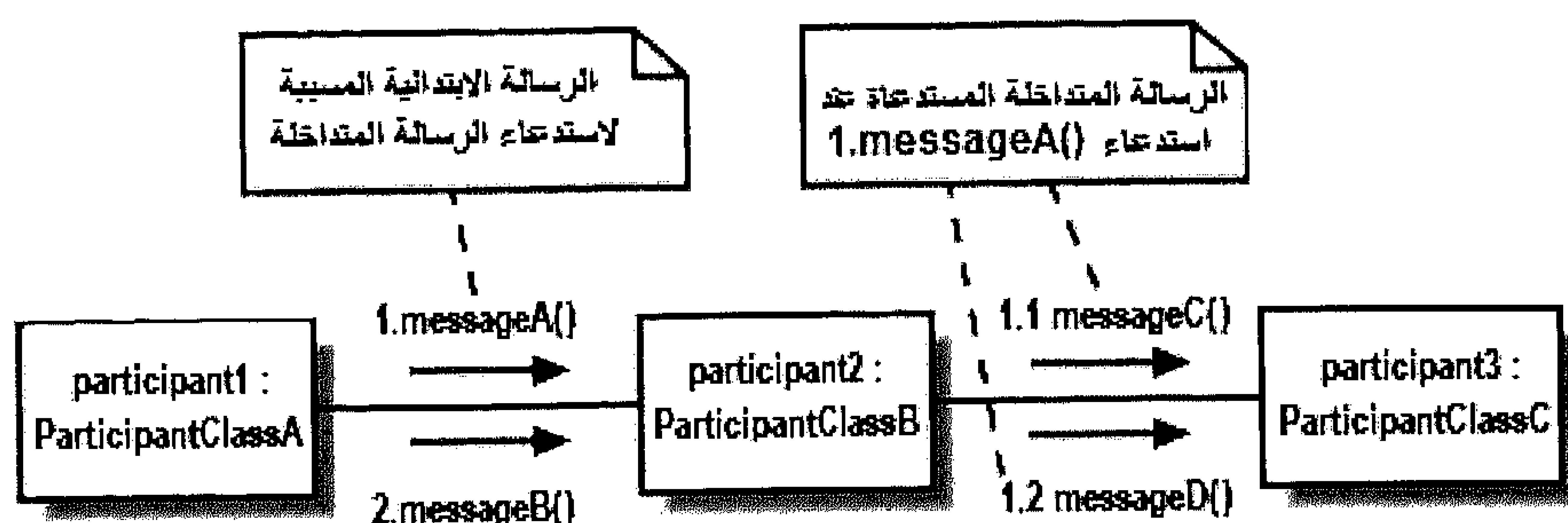
وليس من الضروري تدفق مخططات الاتصال نحو أسفل الصفحة مثل مخططات التتابع؛ لذلك، يتم إظهار رتبة الرسالة في مخطط الاتصال باستعمال رقم قبل كل رسالة. ويشير رقم الرسالة إلى الرتبة التي تستدعى بها الرسالة، ويبدأ الترقيم من الرقم ١ و بشكل تصاعدي حتى ترقيم كل رسائل المخطط. وبإتباع هذه القاعدة في الشكل رقم (٨-٢)، يتم أولاً استدعاء الرسالة 1.messageA() ومن ثم استدعاء الرسالة 2.messageB().

وتصبح الأمور أكثر تعقيداً عندما تتسبب رسالة مرسلة إلى مشارك ما بالقيام مباشرة بإرسال رسالة أخرى منه. عندما تتسبب رسالة ما بإرسال رسالة أخرى، يقال للرسالة الثانية أنها داخلية أو متداخلة nested داخل الرسالة الأصلية، كما هو معروض في الشكل رقم (٨-٣).



شكل رقم (٨-٣) من السهل ملاحظة الرسائل المتداخلة في مخطط التتابع؛ عندما يتم استدعاء الرسالة الأولى message1(..) من خلال المشارك participant2، ثم يتم استدعاء المشارك participant2 للرسالة المتداخلة message2(..) من خلال المشارك participant3.

تستعمل مخططات الاتصال طريقة ترقيم لرسائلها لعرض ترتيب الرسائل المتداخلة (عدة مستويات في الترقيم لتبيان تداخل الرسائل). وإذا قلنا أن رقم الرسالة الأولى هو .1، تبدأ بالتالي أرقام الرسائل المتداخلة داخل الرسالة الأولى بالجزء .1 ثم يضاف رقم عند نهاية الجزء بعد النقطة لترتيب الرسائل المتداخلة. وإذا كان رقم الرسالة الأولى هو الرقم .1، يكون بالتالي رقم الرسالة المتداخلة الأولى هو الرقم .1.1. ويعرض الشكل رقم (٨-٤) مثالاً لهذا الترتيب للرسائل المتداخلة.

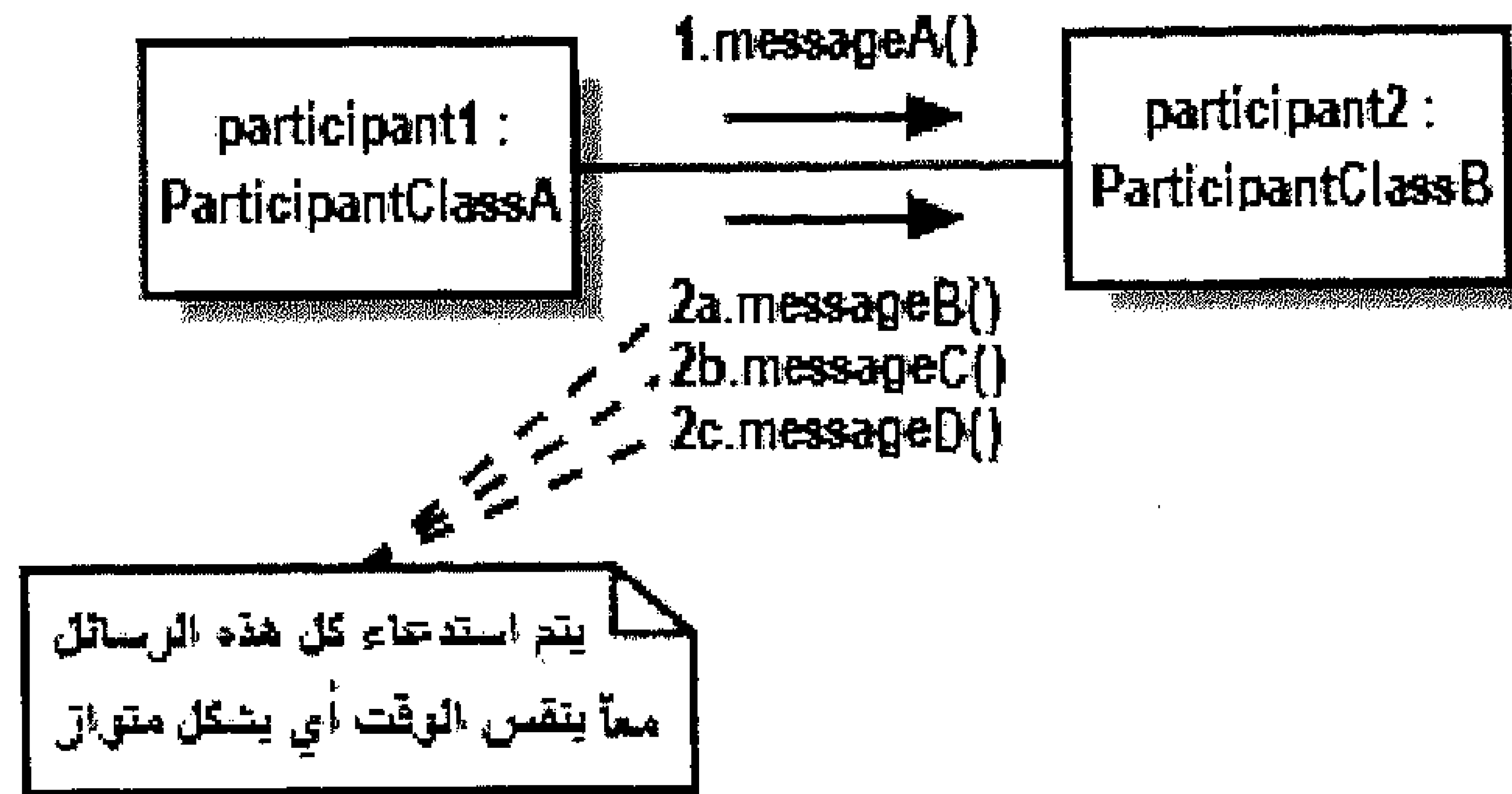


شكل رقم (٨-٤) تؤدي 1.messageA() مباشرة إلى الرسالة المتداخلة 1.1.messageC()، ثم تتبع بالرسالة المتداخلة 1.2.messageD()، قبل استدعاء الرسالة 2.messageB().

٨-١-١ الرسائل التي تحدث في نفس الوقت

Messages Occurring at the Same Time

توفر مخططات الاتصال جواباً بسيطاً لمشكلة الرسائل التي يتم استدعاؤها بنفس الوقت. وبالرغم من حاجة مخططات التتابع إلى بنية معقدة، مثل أقسام التتابع المتوازية par، تستغل مخططات الاتصال ترتيب الرسائل التابعة لها المبني على الترقيم بإضافة ترميز الرقم والحرف **number-and-letter**، وذلك للإشارة إلى حدوث عدة رسائل بنفس الوقت، كما هو معروض في الشكل رقم (٨-٥).



شكل رقم (٨-٥) يتم استدعاء كل الرسائل 2a.messageB()، 2b.messageC()، و2c.messageD() جميعاً بنفس الوقت بعد استدعاء الرسالة 1.messageA().

٨-١-٢ استدعاء رسالة عدة مرات

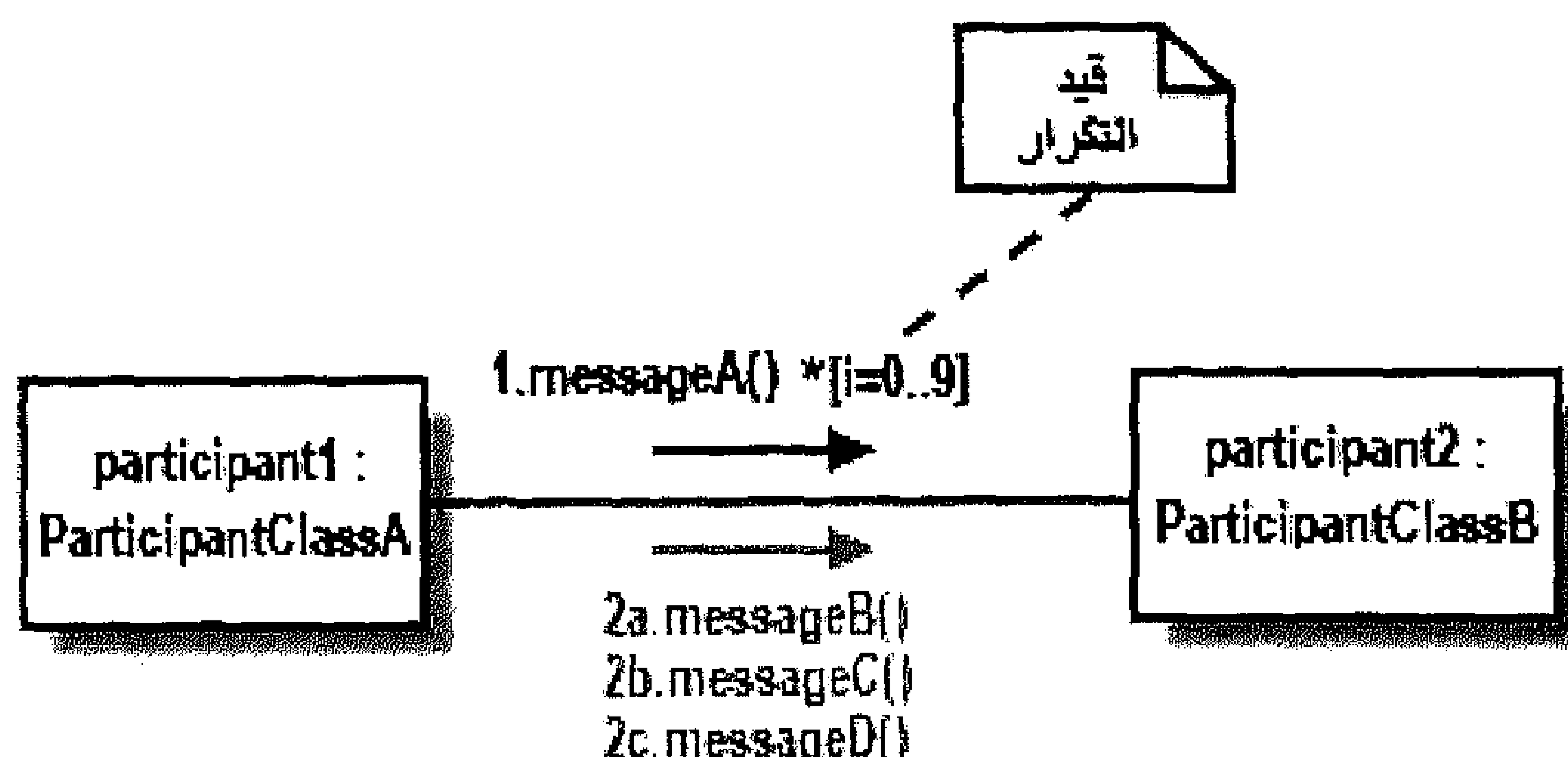
Invoking a Message Multiple Times

عند وصف الرسائل في مخطط الاتصال، ربما تريد إظهار استدعاء رسالة ما عدداً من المرات. يشبه هذا إظهار استدعاء رسائلك داخل تكرار (..) for في حال كنت تقوم بإسقاط مشاركي مخطط الاتصال على البرامج.

وبالرغم من عدم تحديد لغة النمذجة الموحدة حالياً كيف يمكن لمخطط الاتصال إظهار استدعاء رسالة ما عدداً من المرات، لكنها تعلن وجوب استعمال رمز النجمة * قبل تطبيق أي قيد تكرار. يعني هذا التصريح المعقد نسبياً أن المثال التالي هو وسيلة آمنة لتحديد حدوث شيء ما ١٠ مرات:

*[i = 0.. 9]

في مثال قيد التكرار السابق، يمثل المتغير i عداداً يقوم بالعد من الرقم ٠ إلى الرقم ٩، وينفذ ١٠ مرات أية مهمة مرافقة له. ويظهر الشكل رقم (٨-٦) كيفية تطبيق قيد التكرار السابق على رسالة في مخطط الاتصال.



شكل رقم (٨-٦) تعني إضافة قيد التكرار الجديد إلى الرسالة 1.messageA() أنه سيتم استدعاء هذه الرسالة ١٠ مرات قبل أن يصبح بالاستطاعة استدعاء المجموعة التالية من الرسائل 2a، 2b و 2c.

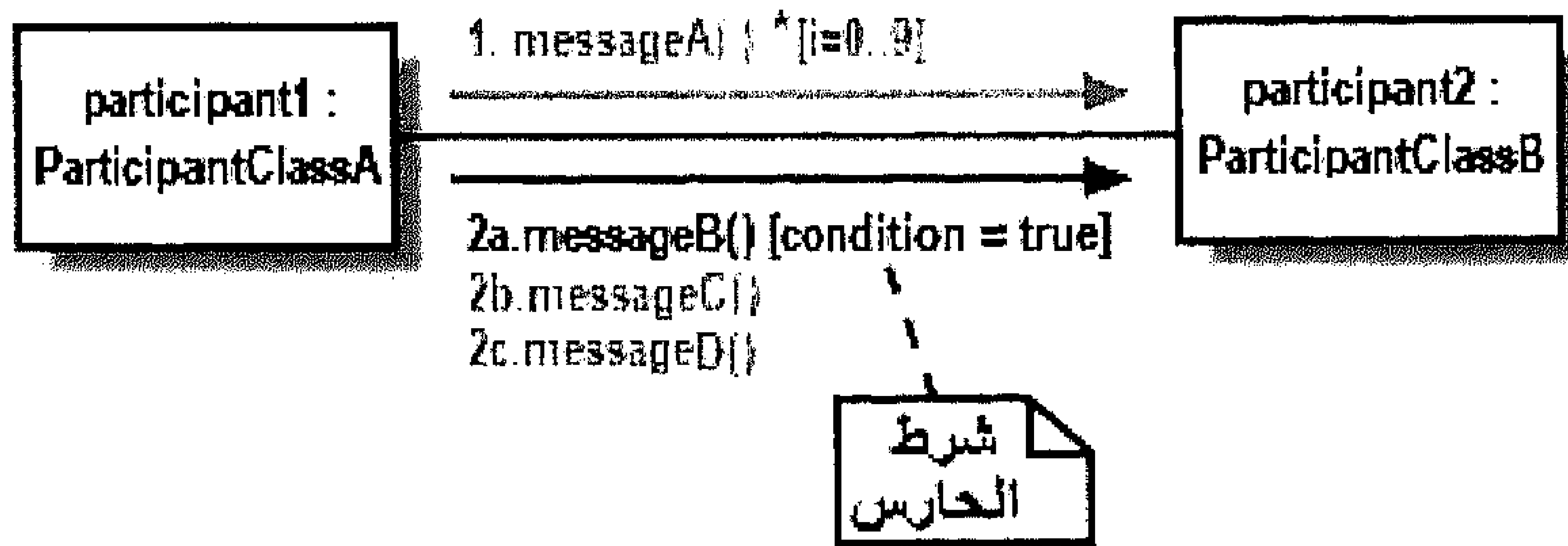
٨-١-٣ إرسال رسالة بالارتكاز على شرط ما

Sending a Message Based on a Condition

يجب أحياناً إرسال رسالة عندما تكون قيمة شرط معيّن صحيحة فقط. على سبيل المثال، يمكن أن يكون في النظام رسالة ما يجب استدعاؤها فقط إذا تم تنفيذ الرسالة السابقة بشكل صحيح. وكما هو الحال ببساطة مع أقسام التتابع في مخطط التتابع، ويمكن أن يكون لرسائل مخطط الاتصال حراس مهيّئون لوصف الشروط المطلوب تقييمها قبل تنفيذها.

يتألف شرط الحارس **guard condition** من تعبير منطقي. إذا تم تقييم شرط الحارس على القيمة صح، سيتم إرسال الرسالة المرافقة له وإلا فيتم تجاوزها.

يعرض الشكل رقم (٨-٧) كيفية تطبيق شرط الحارس على رسالة من بين ثلاثة رسائل تنفذ بشكل متوازٍ.

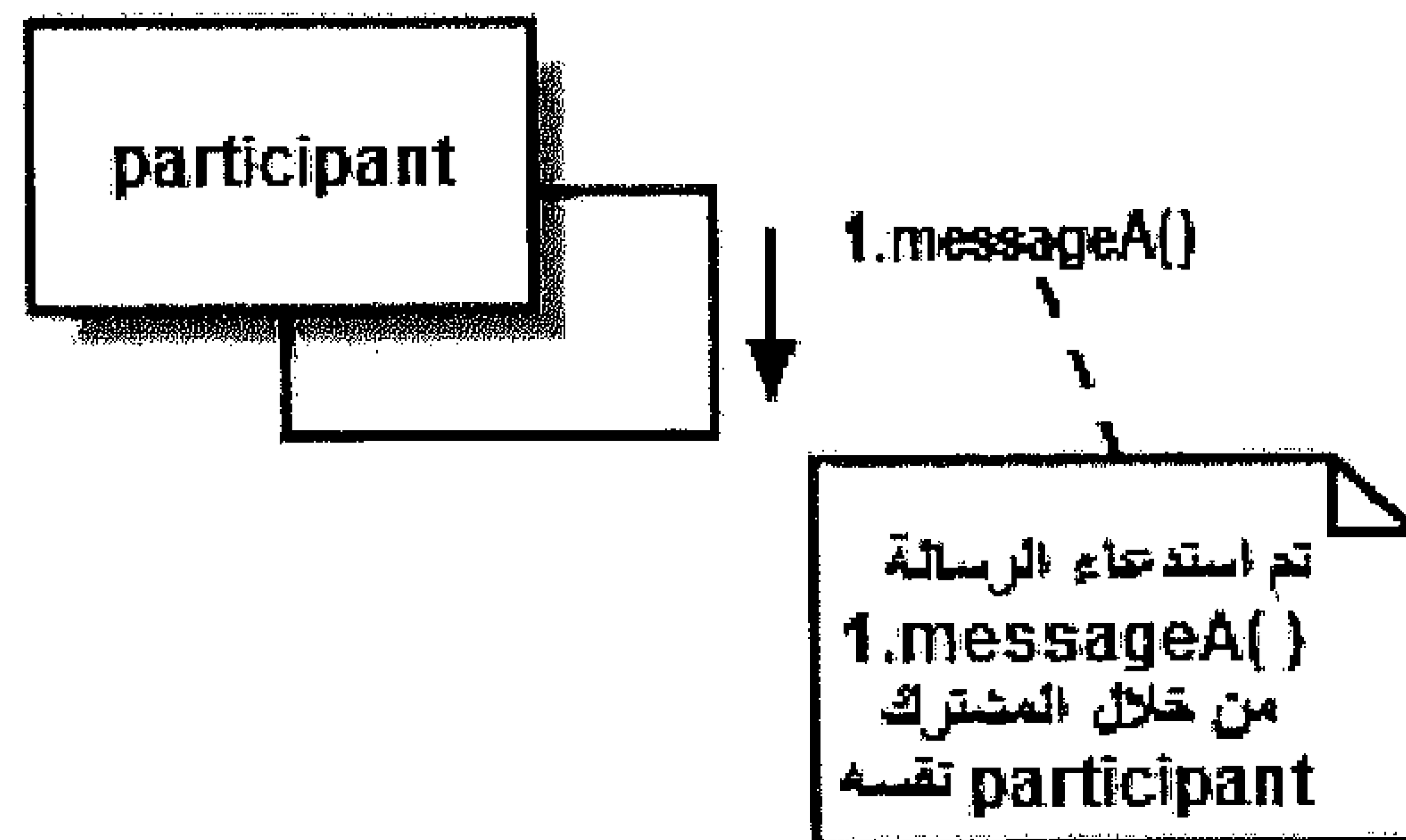


شكل رقم (٨-٧) سيتم إرسال الرسالة 2a.messageB() في نفس وقت إرسال الرسالتين 2b.messageC() و 2c.messageD() في حال تقييم الشرط condition = true على القيمة صح فقط؛ لن ترسل الرسالة 2a.messageB() إذا كانت قيمة هذا الشرط خطأ ولكن سيتم إرسال الرسالتين 2b.messageC() و2c.messageD().

٨-١-٤ عندما يرسل مشارك رسالة لنفسه

When a Participant Sends a Message to Itself

يمكن بداية أن يعطي القول أن المشارك يتكلم مع نفسه انطباعاً غريبا، لكن إذا فكرّت من ناحية قيام كائن برمجي باستدعاء إحدى طرقه، فربما تبدأ تتصور ضرورة هذا الشكل من الاتصال (وحتى رواجه). ويستطيع المشارك في مخطط الاتصال أن يرسل رسالة لنفسه، كما هو الحال ببساطة مع مخططات التتابع. وكل المطلوب هنا هو مجرد رابط من المشارك إلى نفسه للسماح بإرسال الرسالة، كما هو معروض في الشكل رقم (٨-٨).

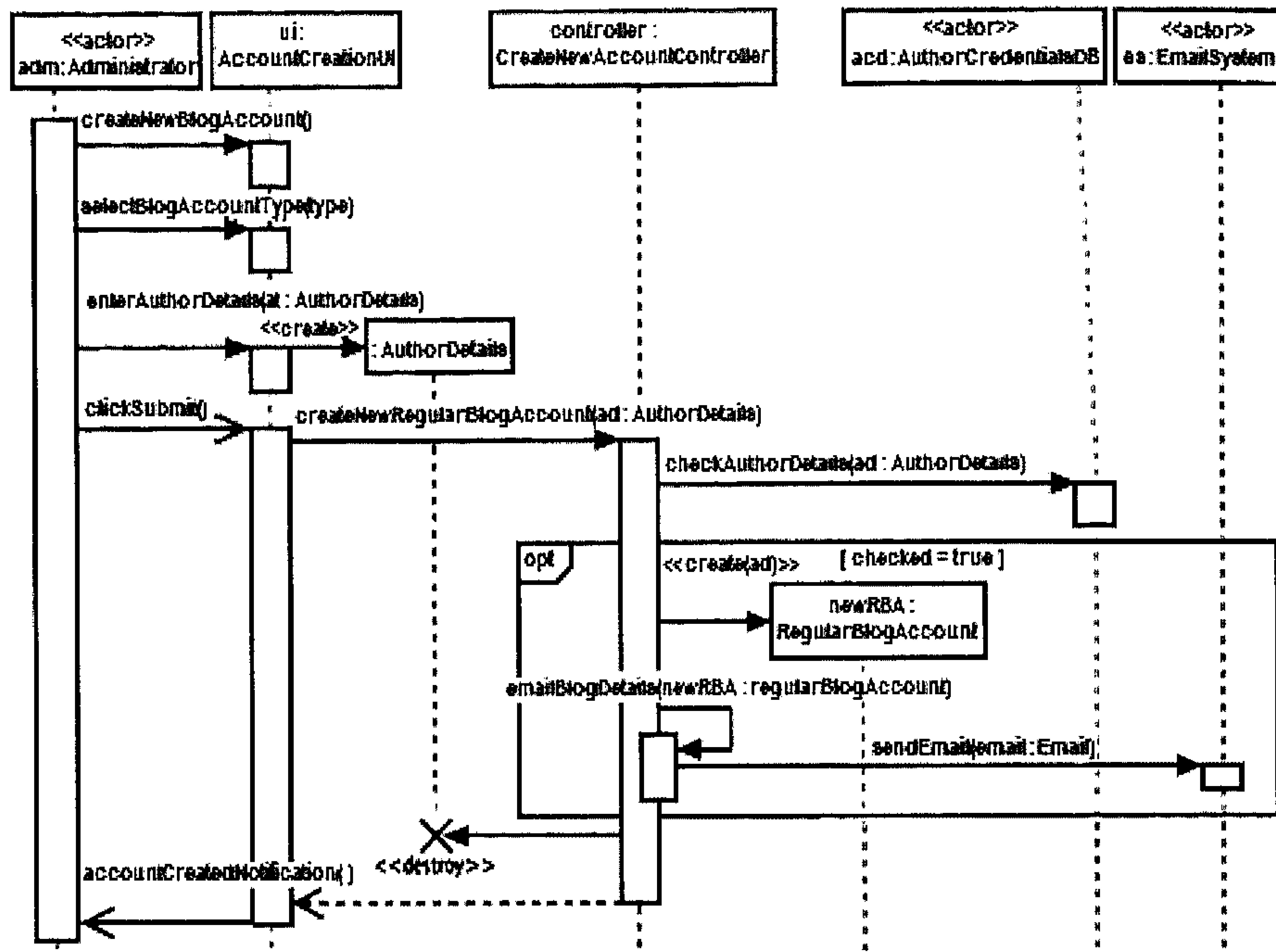


شكل رقم (٨-٨) يمكن للمشارك participant إرسال الرسالة 1.messageA() إلى نفسه بسبب امتلاكه رابط اتصال مع نفسه.

٨-٢ إضافة تفاصيل لتفاعلٍ ما مع مخطط اتصال

Fleshing out an Interaction with a Communication Diagram

حان الوقت الآن للنظر في مثال عملي مع الترميز الجديد لمخطط الاتصال. سنأخذ أحد مخططات التتابع من الفصل السابع ونعرض أيضاً كيفية نمذجة تفاعلاتها على مخطط الاتصال انظر الشكل رقم (٨-٩).

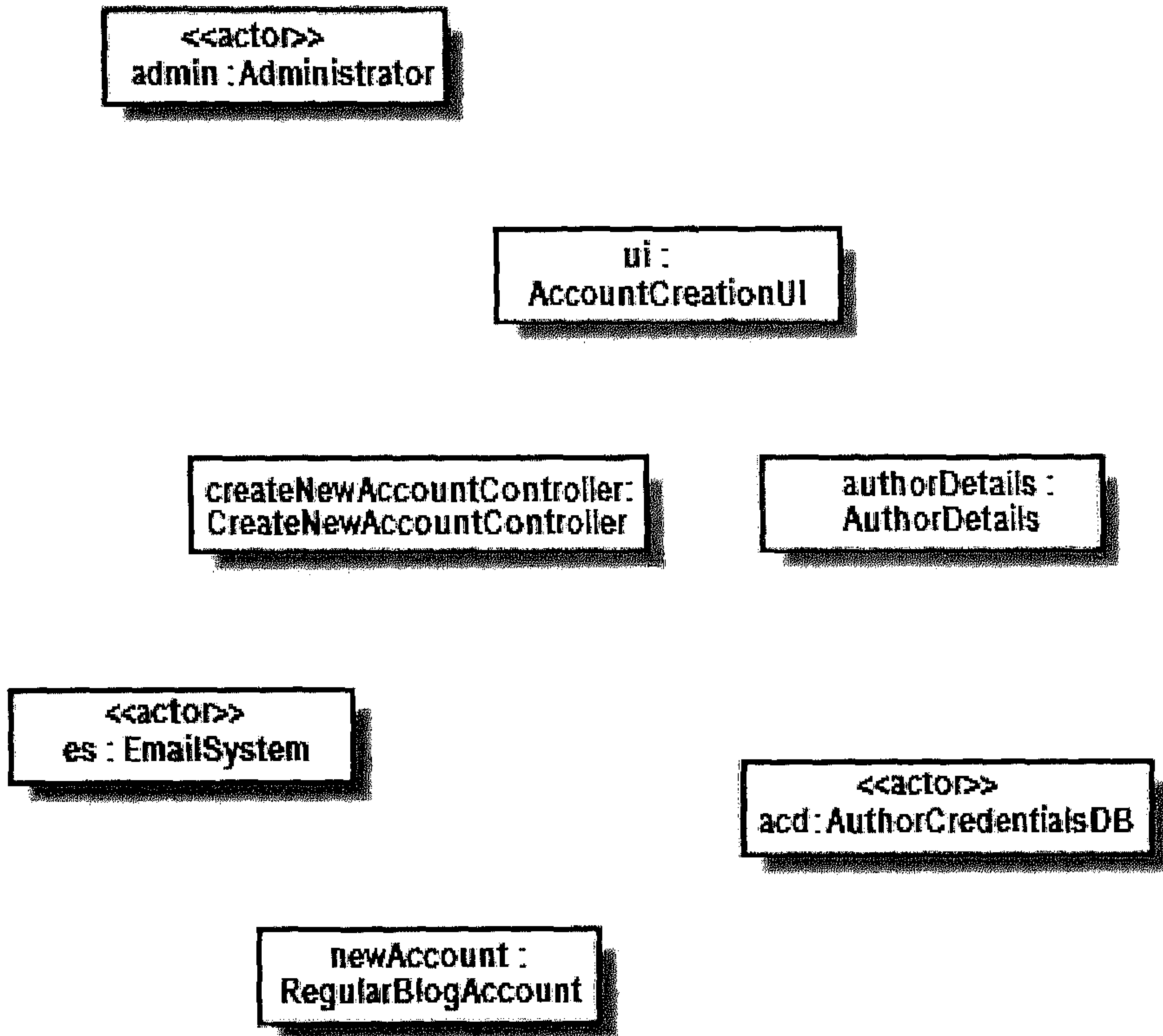


شكل رقم (٨-٩) يصف مخطط التتابع التفاعلات التي تحدث عند إنشاء حساب مدونة عادي جديد في نظام إدارة المحتوى CMS.

لا تقلق من الرجوع إلى الفصل السابع لمساعدتك بخصوص الترميز المعروض في مخطط التتابع. تحتوي مخططات التتابع على كثير من

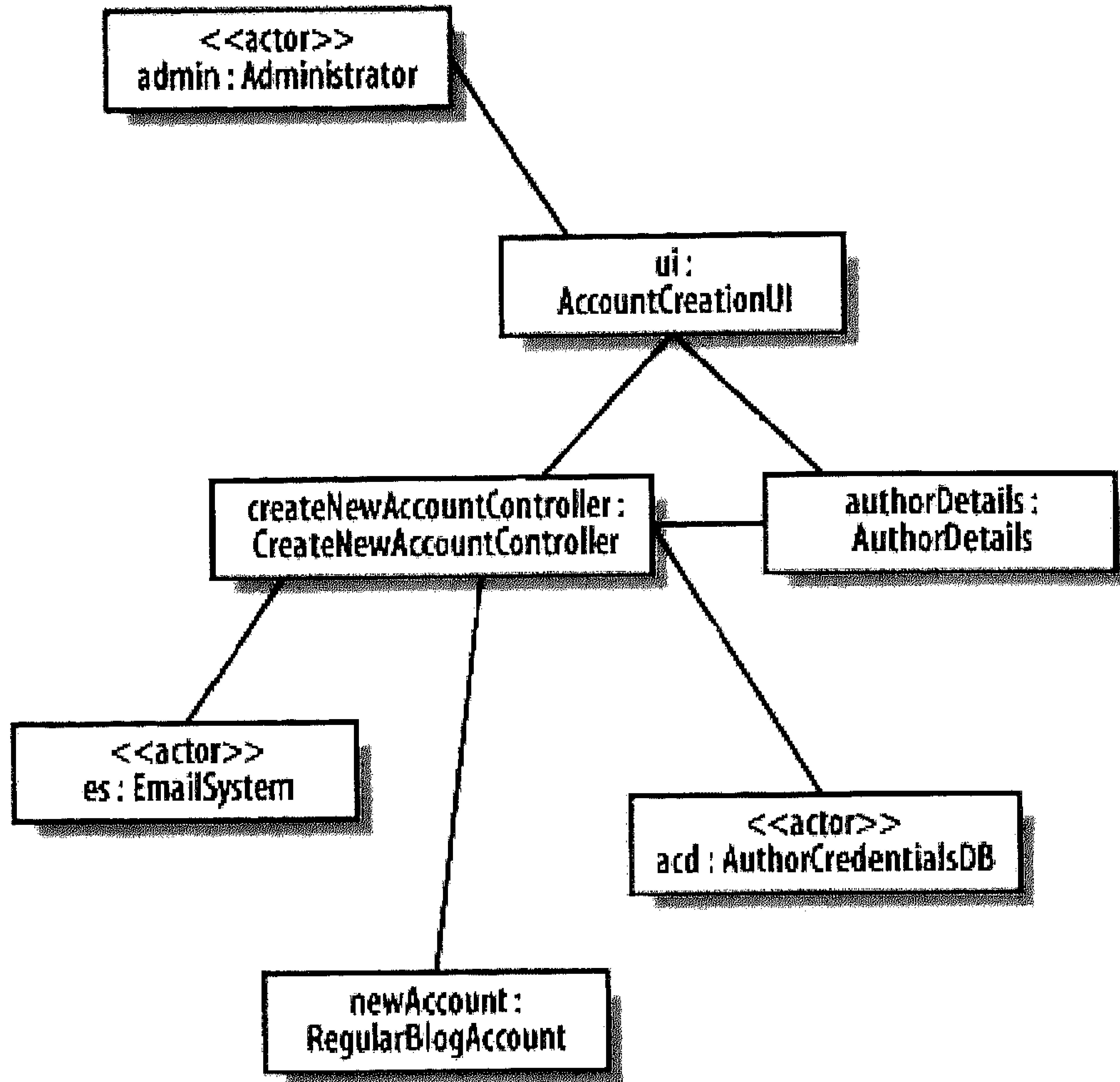
الترميزات الفريدة التي يستغرق إتقانها بعض الوقت! وليس من الضروري تواجد مخطط التتابع قبل إنشاء مخطط الاتصال. ويمكن إنشاء مخطط الاتصال أو مخطط التتابع للتفاعلات بالترتيب الذي تراه مناسباً.

نبدأ أولاً بإضافة المشاركين من الشكل رقم (٨-٩) إلى مخطط الاتصال المعروض في الشكل رقم (٨-١٠).



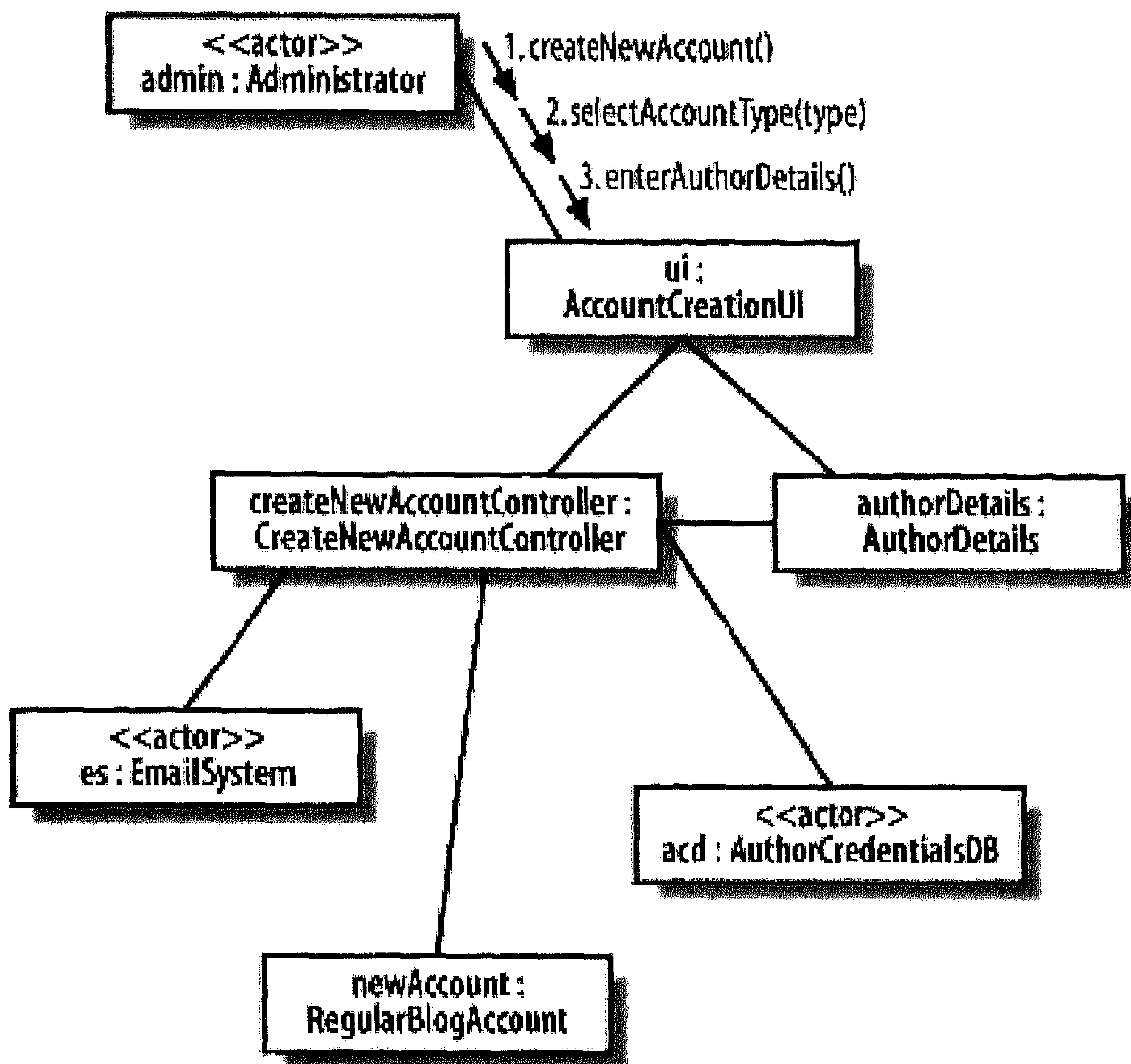
شكل رقم (٨-١٠) عادة ما يكون المشاركين المتعلقين بالتفاعل أول العناصر المضافة إلى مخطط الاتصال.

يتم بعد ذلك إضافة الروابط التي بين المشاركين ليتمكنوا من الاتصال فيما بينهم، كما هو معروض في الشكل رقم (٨-١١).



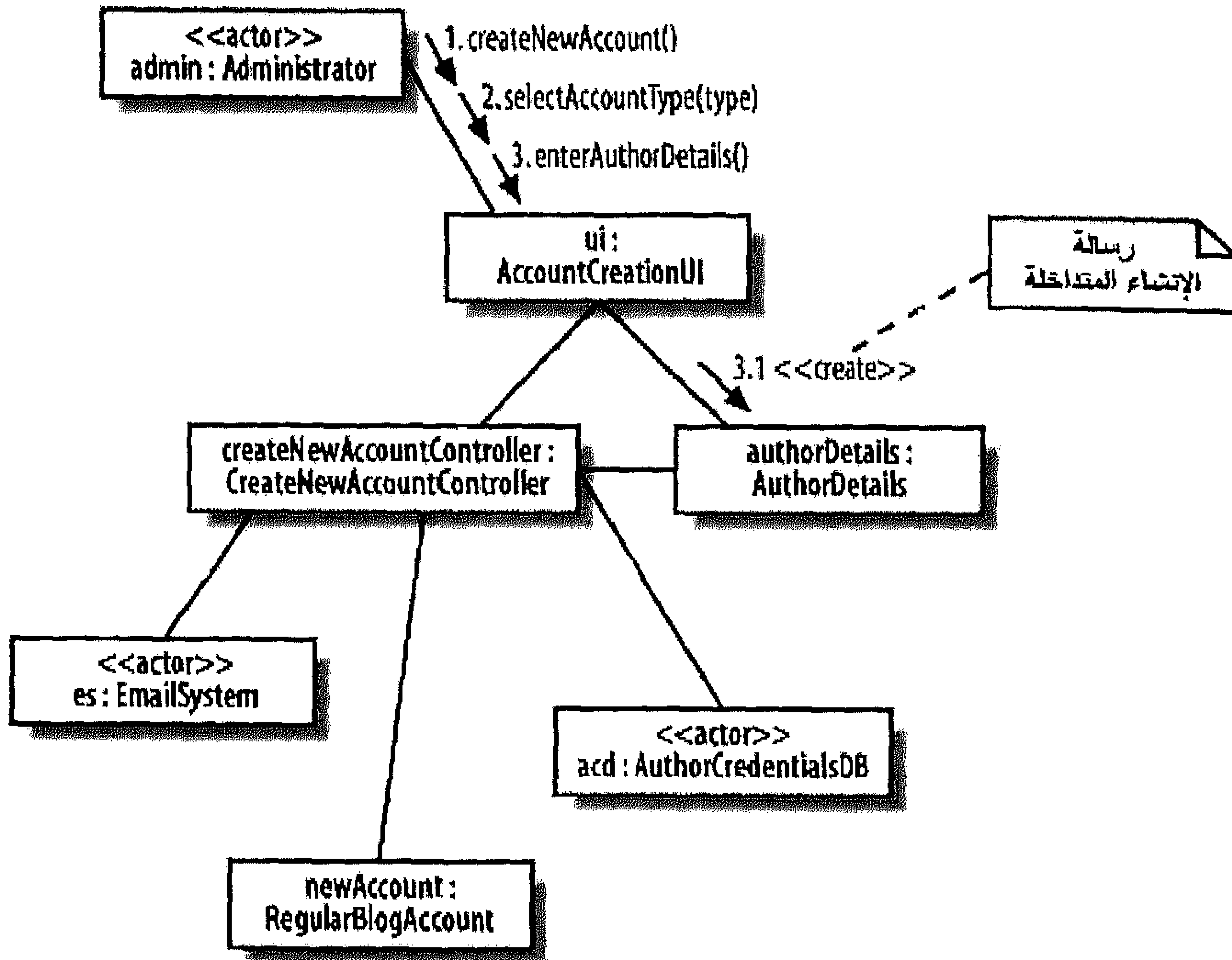
شكل رقم (٨-١١) بالنظر إلى مخطط التتابع في الشكل رقم (٨- ٩)، يمكن إضافة الروابط المتطلبة لدعم الرسالة الممررة في مخطط الاتصال.

حان الوقت الآن لإضافة الرسائل المرسلة بين المشاركين أثناء حياة التفاعل، كما هو معروض في الشكل رقم (٨-١٢). عند إضافة رسائل إلى مخطط الاتصال، عادة ما يفضل البدء مع المشارك أو الحدث الذي يفتتح التفاعل.



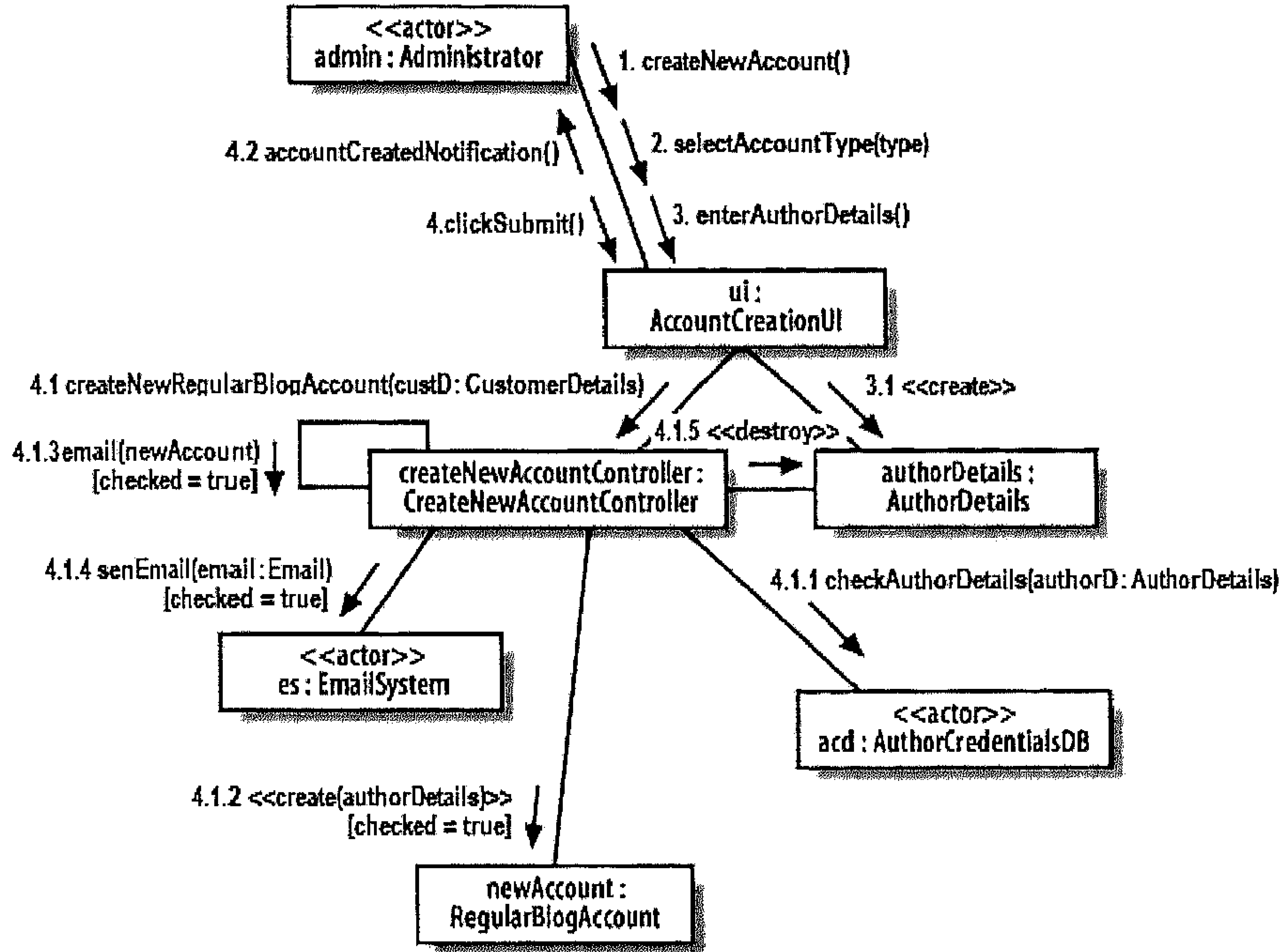
شكل رقم (٨-١٢) يفتتح المستخدم **Administrator** الأمور بتمرير ثلاث رسائل منفصلة إلى المشارك **ui: AccountCreationUserInterface**.

بعد إضافة الرسالة أو الرسائل الاستهلالية إلى مخطط الاتصال، تبدأ الأمور بالتعقيد. وتطلق الرسالة 3.enterAuthorDetails() رسالة إنشاء داخلية من المشارك ui: AccountCreationUserInterface لإنشاء مشارك جديد authorDetails: CustomerDetails. وتأخذ الرسائل المتداخلة فاصلة عشرية إضافية (نقطة) بالاستناد إلى رقم الرسالة المُطلِقة، كما هو معروض في الشكل رقم (٨-١٣).



شكل (٨-١٣) عندما تتم إضافة الرسالة <<create>> إلى مخطط الاتصال، يوضع رقم ترتيبها على 3.1 لإظهار أنها داخلية داخل الرسالة 3.enterAuthorDetails().

مع تلك العقبة الصغيرة التي لم تعد مشكلة، يمكن إضافة بقية الرسائل إلى مخطط الاتصال، انظر إلى الشكل رقم (٨-١٤).



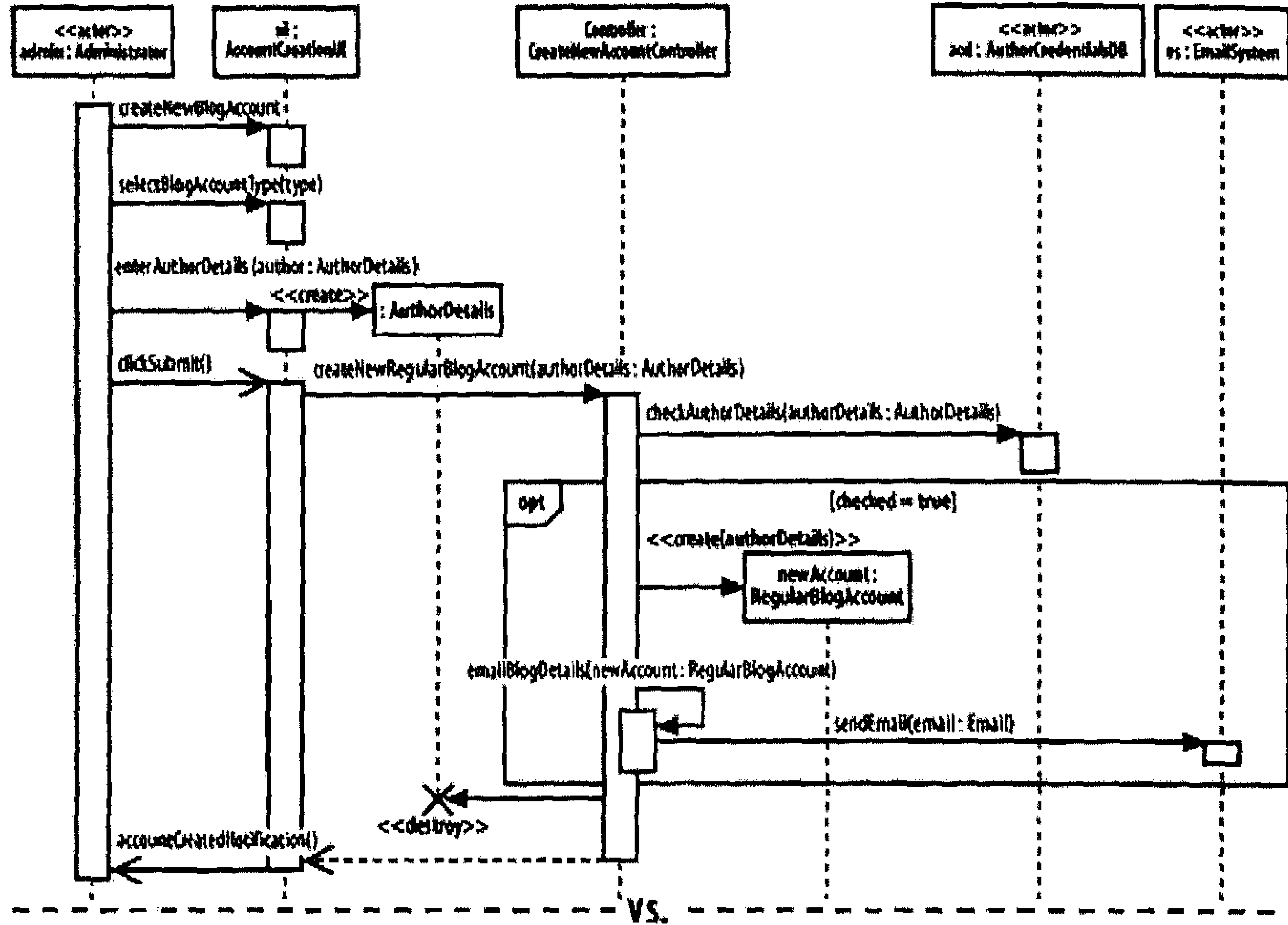
شكل (٨-١٤) يعرض مخطط الاتصال النهائي كامل مجموعة الرسائل ضمن التفاعل "إنشاء حساب مدونة عادي جديد" وفقاً للرسائل المعروضة في مخطط التتابع الأصلي المعروض في الشكل (٨- ٩).

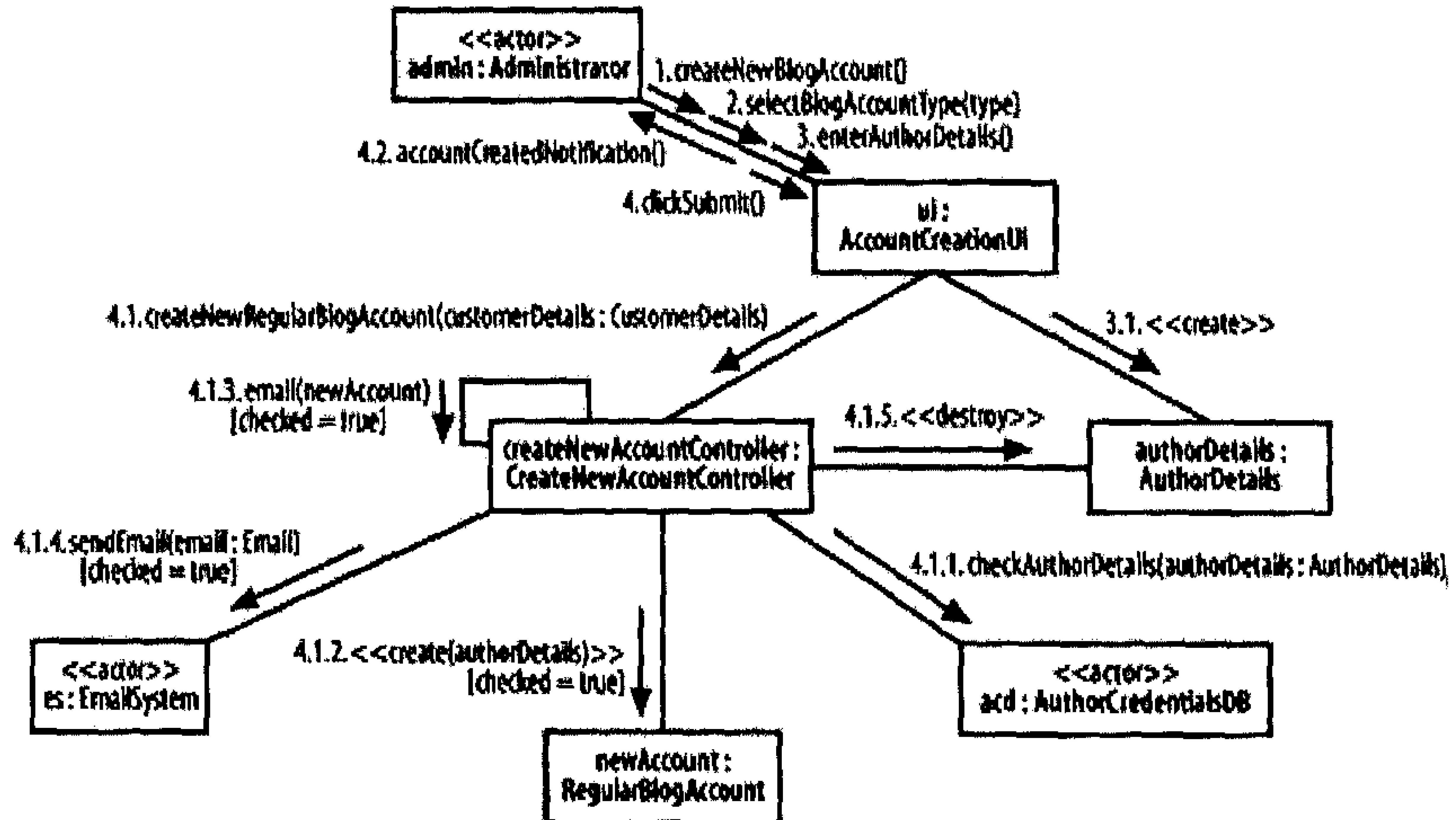
٨-٣ مخططات الاتصال مقابل مخططات التتابع
Communication Diagrams versus Sequence Diagrams

تعرض مخططات الاتصال ومخططات التتابع معلومات متماثلة مما يجعل المقارنة بينهما أمراً حتمياً. لنضع التفضيلات الشخصية جانبا، ما هي أفضل الأسباب لاختيار مخطط تتابع، أو مخطط اتصال، أو حتى خليطاً منهما معاً لنمذجة تفاعل محدد؟

٨-٣-١ كيف يتطور الصراع How the Fight Shapes Up؟

يعرض الشكل رقم (٨-١٥) التمثيلين المختلفين لنفس التفاعل "إنشاء حساب مدونة عادي جديد".





شكل رقم (٨-١٥) يمكن نمذجة "إنشاء حساب مدونة عادي جديد" باستخدتم مخطط التتابع الذي بالأعلى (نفس الشكل ٨- ٩) أو باستخدام مخطط الاتصال الذي بالأسفل (نفس الشكل ٨- ١٤).

٨-٣-٢ الحدث الرئيسي The Main Event

بعيداً عن أي حجج حول التفضيل الشخصي و باستعمال التفاعل المعروض في الشكل رقم (٨-١٥) كمثال، يقارن الجدول رقم (٨-١) بين مخططات التتابع ومخططات الاتصال للمساعدت في اتخاذ القرار بشأن أي منهما يكون الأكثر إفادة لأجل أهداف للنمذجة.

جدول رقم (٨-١) مقارنة بين مخططات التتابع و مخططات الاتصال.

الميزة	مخططات التتابع	مخططات الاتصال	النتيجة
عرض المشاركين بفعالية	يرتب أغلب المشاركين في موازاة أعلى الصفحة، إلا عند استعمال ترميز صندوق إنشاء المشارك المنخفض	يكون التركيز على المشاركين بالإضافة إلى الروابط، لذلك يتم عرضها بشكل واضح مثل المستطيلات.	بالكاد تفوز مخططات الاتصال. بالرغم من استطاعة نوعي المخططات عرض المشاركين بنفس الفعالية، بالإمكان الإدعاء أن مخططات الاتصال لها الأفضلية لأن المشاركين هم أحد تركيزاتها الاساسية.
عرض الروابط بين المشاركين	تكون الروابط ضمنية. يدل تمرير رسالة من مشارك إلى آخر إلزامية وجود رابط ضمني بينهم.	تعرض الروابط بين المشاركين بشكل صريح. في الحقيقة، هذا هو الهدف الرئيسي لهذه الأنواع من المخططات.	تفوز مخططات الاتصال لأنها تعرض الروابط بين المشاركين بشكل صريح وواضح.
عرض تواقيع الرسائل	يمكن أن توصف تواقيع الرسائل بالكامل.	يمكن أن توصف تواقيع الرسائل بالكامل.	تعادل! كلا النوعين يعرض الرسائل بنفس الفعالية.
دعم الرسائل المتوازية	تكون مخططات التتابع أفضل بكثير مع إدخال أقسام التتابع.	يتم عرضها باستعمال الترميز رقم - حرف على تتابعات الرسائل.	تعادل! كلا النوعين بإمكانه عرض الرسائل المتوازية جيداً وبشكل متساوي.
دعم الرسائل غير المتزامنة	يتم إنجازها باستعمال سهم عدم التزامن.	ليس لمخططات الاتصال تصور للرسائل غير المتزامنة لأن تركيزها ليس على ترتيب الرسائل.	تفوز هنا مخططات التتابع بشكل واضح لأنها تدعم الرسائل غير المتزامنة بشكل صريح.

الميزة	مخططات التتابع	مخططات الاتصال	النتيجة
سهولة قراءة ترتيب الرسائل	هذا موطن قوة مخططات التتابع فهي تعرض ترتيب الرسائل بوضوح باستعمال الموقع العمودي للرسائل نزولاً لأسفل المخطط.	معروضة باستعمال الترميز مع نقطة الترقيم المتداخل.	تفور هنا مخططات التتابع بشكل واضح لأنها تعرض ترتيب الرسائل بوضوح وفعالية.
سهولة إنشاء وصيانة المخطط	إن إنشاء مخطط تتابع بسيط جداً. على أية حال، يمكن أن تكون صيانة مخطط التتابع كابوساً ما لم يتم استعمال أداة لغة نمذجة موحدة مفيدة.	إن مخططات الاتصال بسيطة كفاية لإنشائها؛ على أية حال، تبقى صيانتها محتاجة للدعم من قبل أداة لغة نمذجة موحدة مفيدة، وذلك بشكل خاص عند الحاجة لتغيير الترقيم	من الصعب الحكم على هذه النقطة وتستند بشكل كبير على التفضيلات الشخصية. على أية حال، لمخططات الاتصال الأفضلية بالارتكاز على سهولة الصيانة.

حسنا، كان الصراع متحيزاً بعض الشيء، حيث كانت الميزات المقيّمة سابقاً عبارة عن مميّز واضح بين مخططات التتابع ومخططات الاتصال. رغم أن النتائج المعروضة في الجدول رقم (٨-١) ليست مفاجئة جداً، لكن من المفيد الإعلان مرة جديدة أنه يجب علينا:

- استعمال مخططات التتابع إذا كنا نهتم بالدرجة الأولى بتدفق الرسائل خلال التفاعل.
- استعمال مخططات الاتصال إذا كنا نركز على الروابط بين مختلف المشاركين المشتركين بالتفاعل.

ربما تكون الرسالة الأكثر أهمية للأخذ بها من خلال هذه المقارنة هي أنه رغم تشابه المعلومات التي تعبر عنها مخططات التتابع

ومخططات الاتصال، لكن تقدم هذه المخططات فوائد مختلفة؛ لذلك، من الأفضل استعمالهما معاً.

٨-٤ ما هي الخطوة التالية؟

لقد رأينا مخططات التتابع و مخططات الاتصال التي تعد أكثر أنواع مخططات التفاعل استعمالاً. إن مخطط التوقيت عبارة عن مخطط تفاعل أكثر تخصصاً حيث يركز على القيود الزمنية الخاصة بالتفاعلات، والتي تكون مفيدة جداً للتعبير عن القيود الزمنية في الأنظمة الفورية real-time. وستتم تغطية مخططات التوقيت في الفصل التاسع.

التركيز على توقيت التفاعل: مخططات التوقيت

FOCUSING ON INTERACTION TIMING: TIMING DIAGRAMS

تدور مخططات التوقيت - من دون اندهاش - حول التوقيت. بينما تركز مخططات التتابع (الفصل السابع) على ترتيب الرسائل، وتعرض مخططات الاتصال (الفصل الثامن) الروابط التي بين المشاركين، ولم يحدد حتى الآن ضمن مخططات التفاعل المذكورة مكان نمذجة المعلومات التفصيلية عن التوقيت. وربما يكون لديك تفاعل يجب إتمامه بأقل من ١٠ ثوان، أو لديك رسالة يجب الرجوع منها بأقل من نصف الوقت الكلي للتفاعل. وإذا كان هذا النوع من المعلومات مهماً بالنسبة للتفاعل الذي تقوم بنمذجته، فمن المرجح أن تكون ضالتك في مخطط التوقيت.

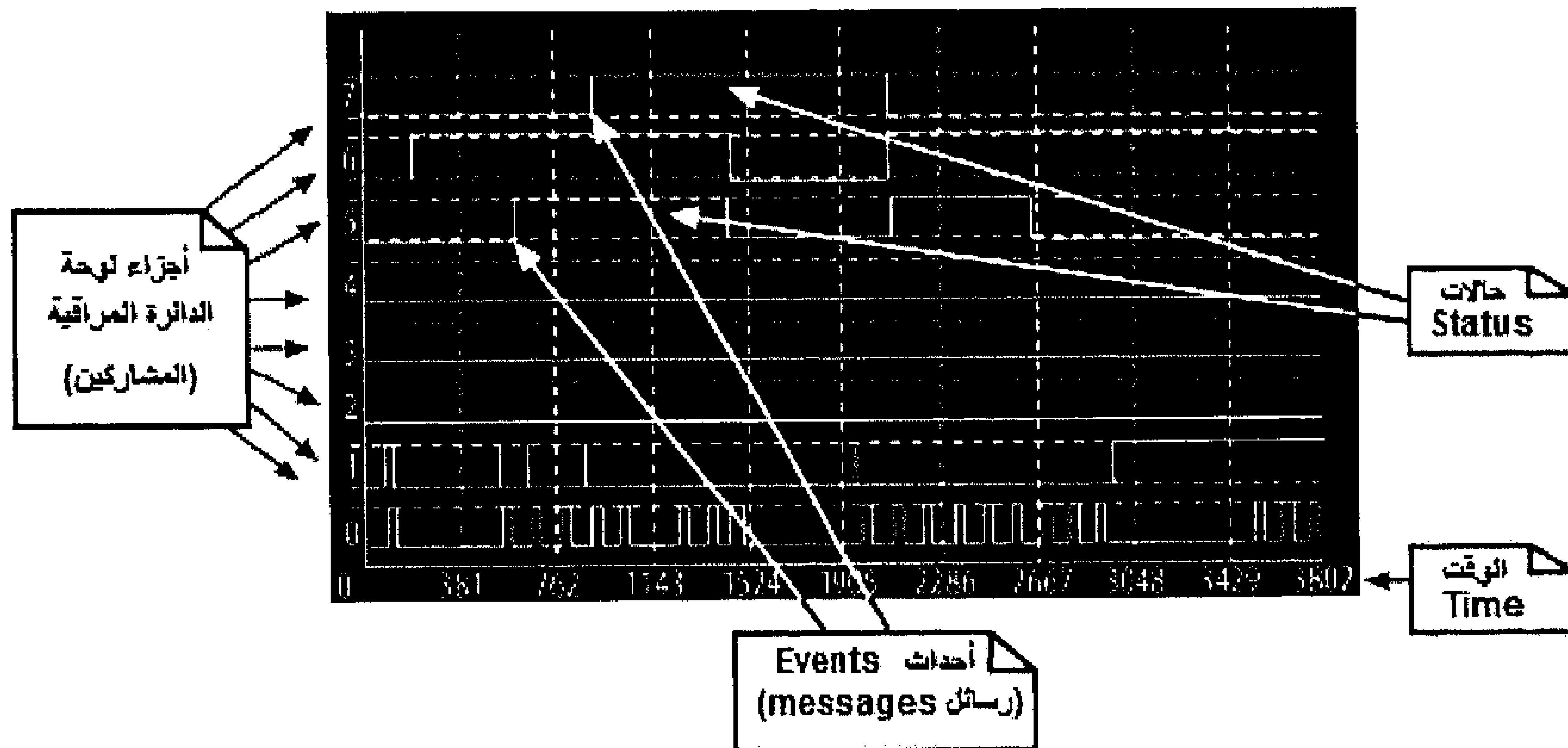
عادة ما يرتبط توقيت التفاعل بالأنظمة الفورية real-time أو بالأنظمة الضمنية embedded، لكنه ليس محصوراً بالتأكيد على هذه المجالات. وفي الحقيقة، يمكن أن تكون الحاجة إلى أسر معلومات توقيت دقيقة حول التفاعل مهمة بغض النظر عن نوع النظام الذي تتم نمذجته.

وفي مخطط التوقيت، يرتبط كل حدث بمعلومات توقيت تصف بدقة وقت انطلاقه، والوقت الذي سيأخذه مشارك آخر لاستلامه،

وكذلك الوقت المتوقع بقاء المشارك المستلم له في حالة محددة. وبالرغم من التشابه الكبير بين مخططات التتابع ومخططات الاتصال، فإن مخطط التوقيت يضيف معلومات جديدة كلياً لم يتم التعبير عنها بسهولة مع أي شكل آخر من مخططات التفاعل في لغة النمذجة الموحدة. ويشبه غياب مخطط توقيت التفاعل القول: "أَعْرِفُ الأحداث التي يجب حدوثها، لكنني لا اهتم حقاً بشأن وقت حدوثها بالضبط أو بسرعة العمل عليها".

٩-١ مظهر مخططات التوقيت
What Do Timing Diagrams Look Like?

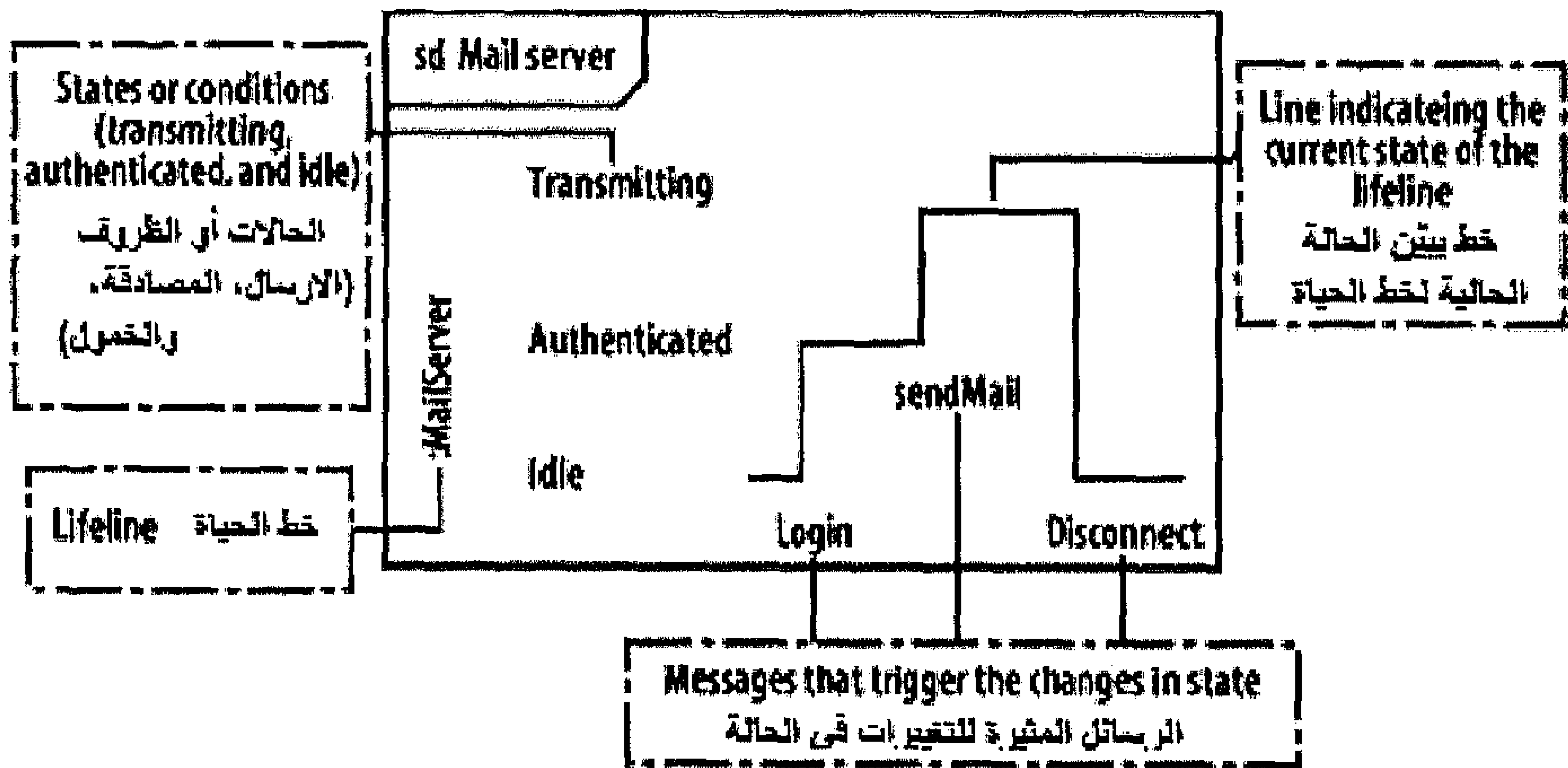
ستبدو مخططات التوقيت مألوفة بشكل استثنائي لقليلي الخبرة في تحليل ألواح الدوائر الكهربائية. هذا بسبب الشبه الكبير بين مخطط التوقيت والخرائط المتوقع رؤيتها على المحلل المنطقي logic analyzer. ولا تقلق إذا لم تشاهد قط محللاً منطقياً من قبل، لذلك؛ يقدم الشكل رقم (٩-١) عرضاً نموذجياً من المتوقع رؤيته على إحدى هذه الأدوات.



شكل رقم (٩-١) تعرض كل المعلومات الظاهرة على المحلل المنطقي على مخطط التوقيت أيضاً على شكل رسائل (events) messages ومشاركين participants وحالات states.

يأسر المحلل المنطقي سلسلة من الأحداث كما تحدث على لوحة دائـرة إلكترونيـة. إن المعلومـات المعروضـة علـى المحلـل المنطقـي (كتلـك المعروضـة في الشـكل رقـم ٩-١)، سـتظهر بشـكل قياسـي الوقـت الـذي تكـون أجـزاء لوحـة الـدائرة المختلفـة عنـده بحالـة محـددة والإشـارات الإلكترونية التي ستطلق التغيير في تلك الحالات.

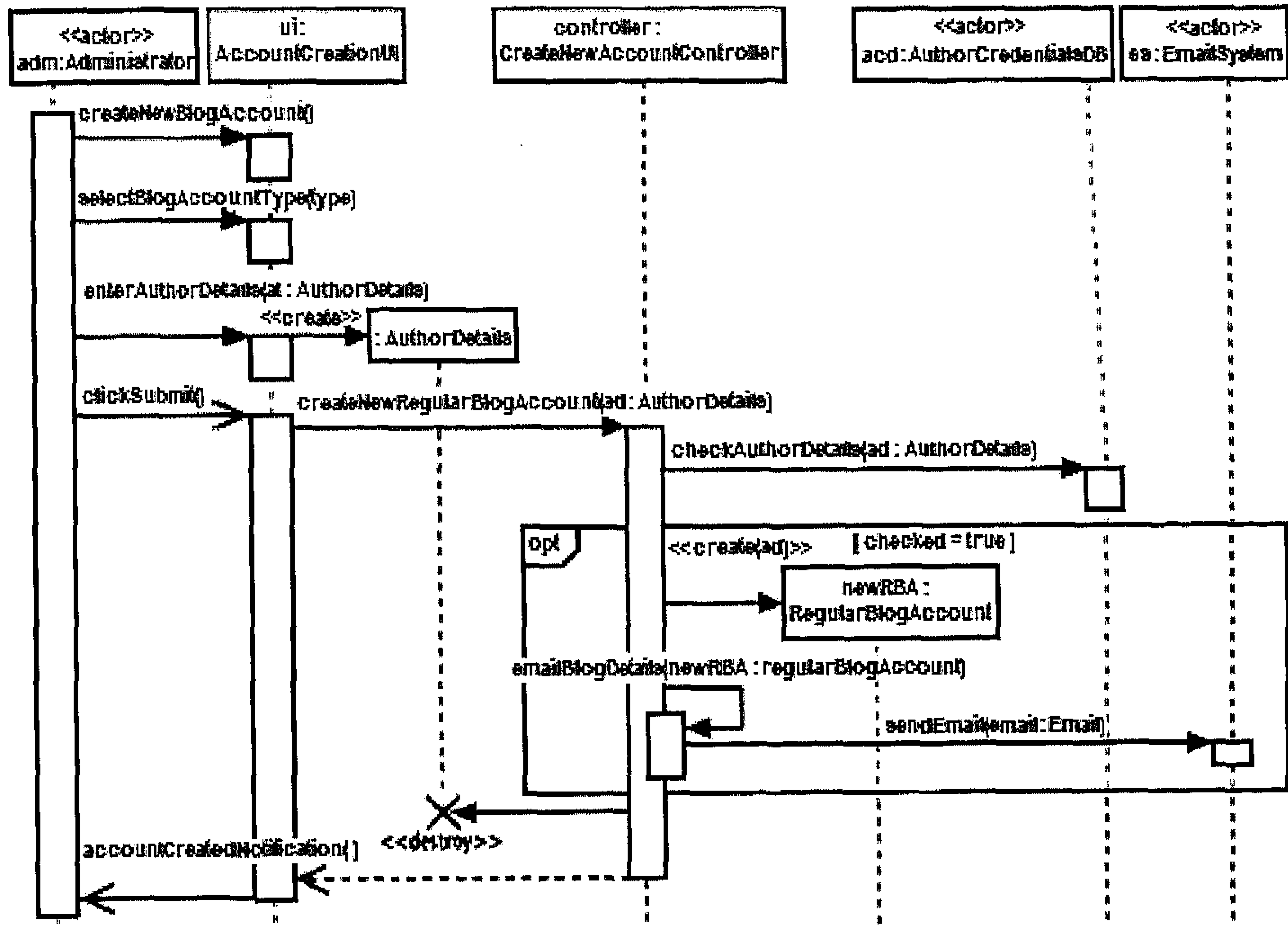
تؤدي مخططات التوقيت عملاً مشابهاً للمشاركين داخل النظام. وتكـون الأحـداث علـى مخطـط التوقيـت، عبـارة عـن إشـارات المحلـل المنطقي، وتكون الحالات عبارة عن الحالات الموضوع المشارك عليها عند استلام حدث ما. وتكون التشابهات بين مخطط التوقيت والمحلل المنطقي ظـاهرة عنـدما تقـارن الشـكل رقـم (٩-١) بالشـكل رقـم (٩-٢)، والـذي يعطي عرضاً أولياً عن شكل مخطط توقيت كامل. ولقد تم اقتباس هذا المثال من الكتاب (UML 2.0 in a Nutshell (O'Reilly.



شكل رقم (٩-٢) قارن مخطط التوقيت البسيط والكامل أيضاً لخادم بريد مع المحلل المنطقي في الشكل رقم (٩-١).

٩-٢ إنشاء مخطط توقيت انطلاقاً من مخطط تتابع
Building a Timing Diagram from a Sequence Diagram

دعنا نجمّع مخطط توقيت من البداية. وسنعمل انطلاقاً من نفس المثال المستعمل في فصول مخططات الاتصال ومخططات التتابع، مثل التفاعل "إنشاء حساب مدونة عادي جديد" المعروض في الشكل رقم (٩-٣).



شكل رقم (٩-٣) يحتوي مخطط التتابع على معلومات قليلة جداً حول التوقيت، هذا إن وجدت، ويكون تركيزها الأساسي حول ترتيب الأحداث داخل التفاعل.

٩-٢-١ قيود التوقيت في متطلبات النظام
Timing Constraints in System Requirements

لقد كان التفاعل المعروض في الشكل رقم (٩-٣) بالأصل نتيجة متطلب ما، مثل ذلك الموصوف في المتطلب أ-٢.

المتطلب أ-٢

يسمح نظام إدارة المحتوى للمدير بإنشاء حساب مدونة عادي جديد، شرط التحقق من تفاصيل الكاتب الشخصية باستعمال قاعدة بيانات اعتماد الكتبة.

دعنا الآن نوسع المتطلب الأولي ببعض الاعتبارات التوقيتية كي يصبح عندنا ما نضيفه على نمذجة التفاعل في مخطط التوقيت.

المتطلب أ-٢ (محدّث)

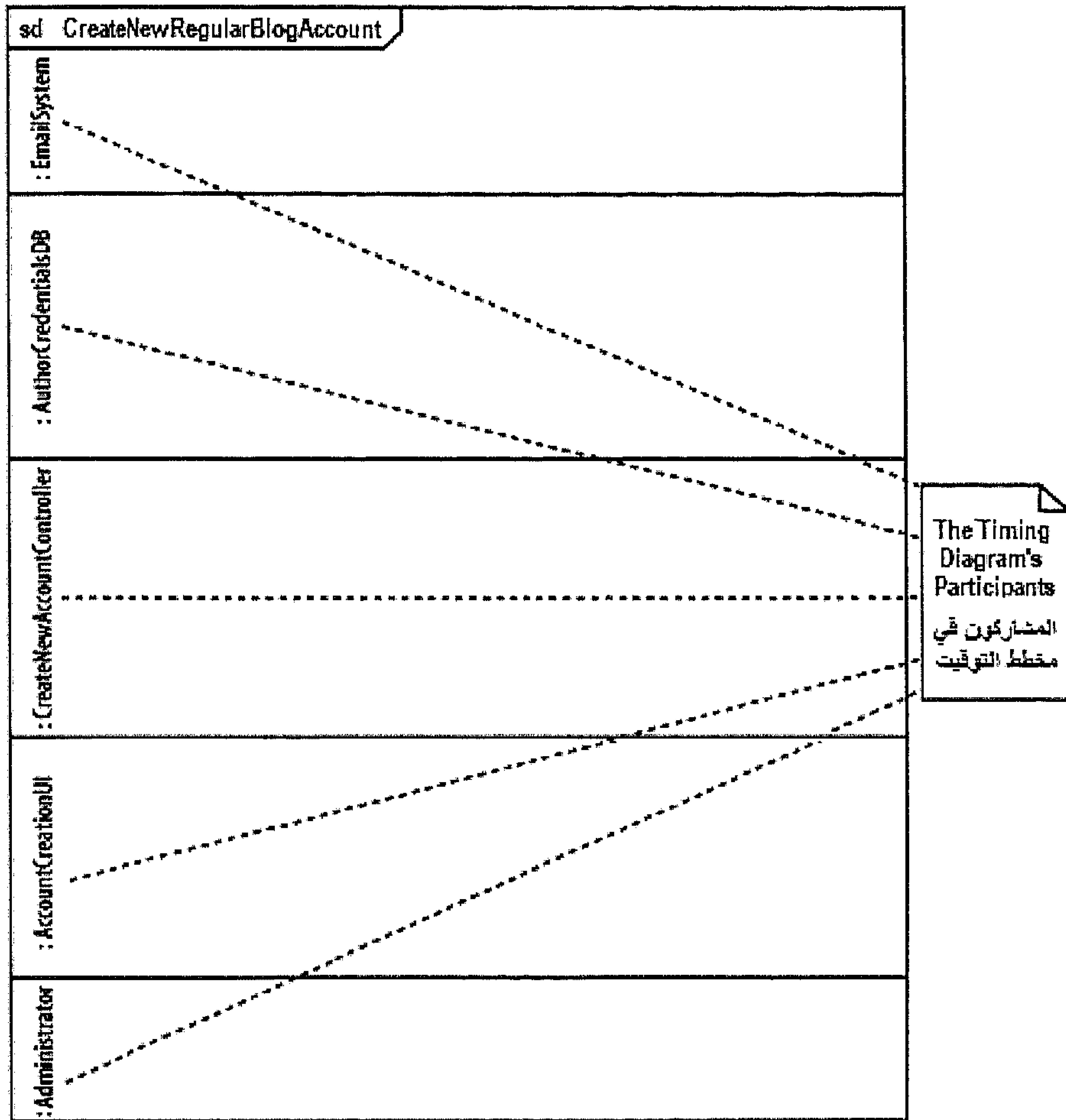
يسمح نظام إدارة المحتوى للمدير بإنشاء حساب مدونة عادي جديد خلال خمسة ثوان من بعد إدخال المعلومات، شرط التحقق من تفاصيل الكاتب الشخصية باستعمال قاعدة بيانات اعتماد الكتبة.

لقد تم تعديل المتطلب أ-٢ لتضمين قيد زمني يحدد المدة التي سيأخذها النظام لقبول والتحقق من وإنشاء حساب جديد. بعد أن أصبح عندنا مزيد من المعلومات حول التوقيت في المتطلب أ-٢، وأصبح لدينا تبرير كاف لنمذجة التفاعل الذي ينجز المتطلب باستعمال مخطط التوقيت.

٩-٣ تطبيق المشاركين على مخطط توقيت

Applying Participants to a Timing Diagram

أولاً، تحتاج إلى إنشاء مخطط توقيت يضم كل المشاركين المشتركين في التفاعل "إنشاء حساب مدونة عادي جديد"، كما هو معروض في الشكل رقم (٩-٤).



شكل رقم (٩-٤) تم كتابة أسماء المشاركين الرئيسيين المعنيين بالتفاعل بشكل عمودي على الجهة التي عن يسار مخطط التوقيت.

لقد تم إهمال الأسماء الكاملة للمشاركين في الشكل رقم (٩-٤)، لإبقاء حجم المخطط سهل الإدارة، رغم أنه باستطاعتك تضمين الصيغة الكاملة لعنوان المشارك باستعمال الصيغة <name>:<type>.

وهناك ميزة أخرى غائبة عن الشكل رقم (٩-٤) و هي المشاركون الذين يتم إنشاؤهم و تدميرهم أثناء حياة التفاعل، مثل المشارك: AuthorDetails

والمشارك: RegularBlogAccount. لقد تم إهمال تفاصيل هذين المشاركَين بسبب تركيز مخططات التوقيت المتعلق بتغييرات الحالة. ليس للمشاركَين: AuthorDetails و RegularBlogAccount أي تغييرات حالة معقدة باستثناء أنه يتم إنشاؤهما و/أو تدميرهما؛ لذلك تم حذفهما لأنهما؛ لا يضيفان شيئاً ذا أهمية لهذا المخطط الخاص.

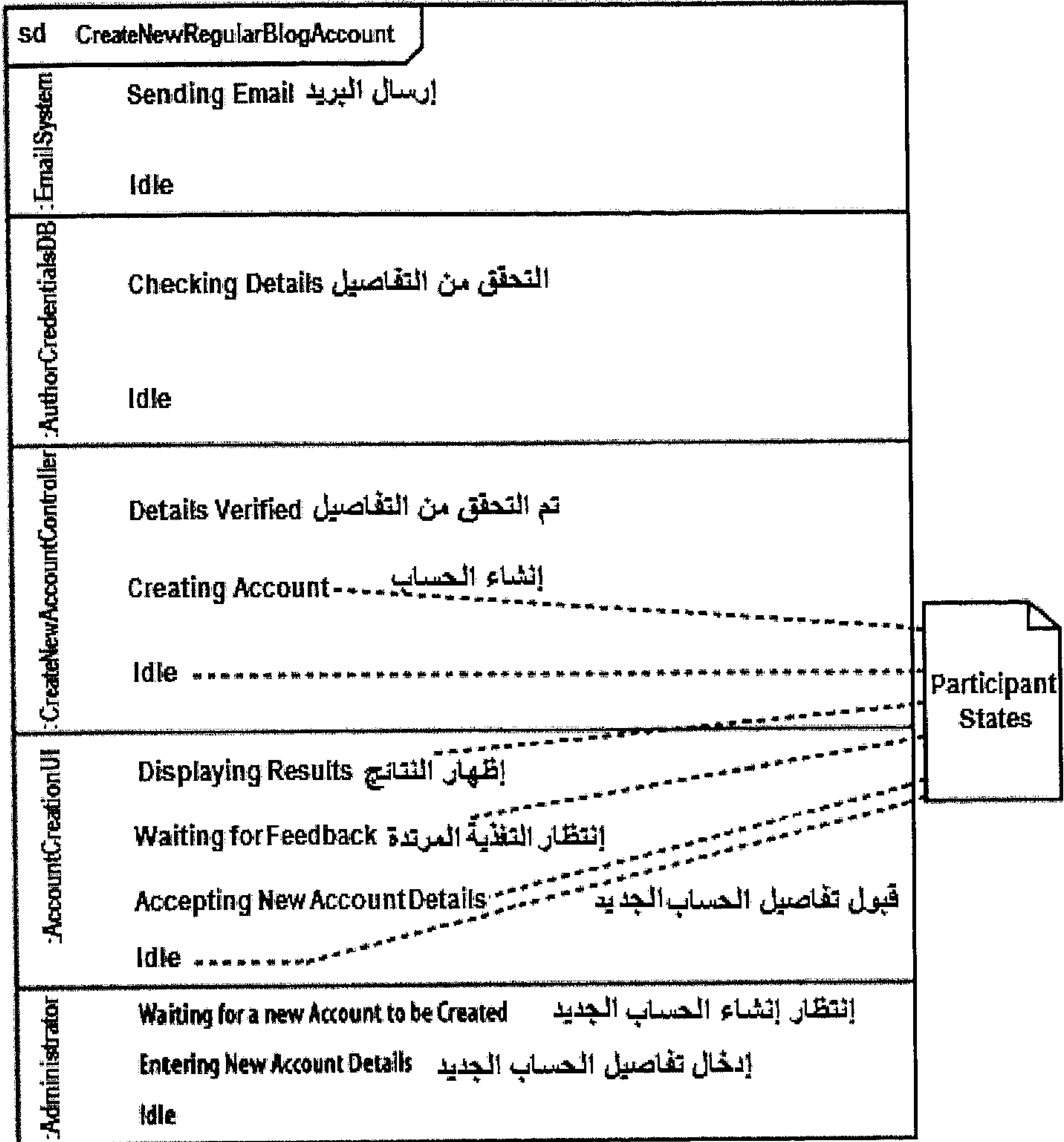


أثناء نشاطات نمذجة النظام، ستحتاج إلى تقرير ما يجب وضعه بشكل صريح على المخطط من عدمه. أسأل نفسك التالي: "هل هذا التفصيل مهم لفهم ما أنمذج؟" و "هل إضافة هذا التفصيل يوضح الأمور أكثر؟"، إذا كان الجواب نعم لأي من هذين السؤالَين، فمن الأفضل إضافة هذا التفصيل في المخطط؛ و إلا فيتم إهماله. ربما تبدو هذه القاعدة فظة نسبياً، إنما يمكن أن تكون فعّالة جداً عندما نريد تقليل الفوضى بالمخطط إلى حدها الأدنى.

٩-٤ الحالات States

يمكن أن يتواجد الكائن أثناء التفاعل بأي عدد من الحالات. ويقال للمشارك بأنه في حالة محددة عند استلامه حدث ما (مثل الرسالة). ويمكن بالتالي القول إن المشارك هو تلك الحالة حتى حصول حدث آخر (مثل الرجوع من تلك الرسالة). انظر إلى قسم "الأحداث و الرسائل" لاحقاً في هذا الفصل لتفسير كيفية تطبيق الأحداث و الرسائل في مخططات التوقيت.

يتم وضع الحالات في مخطط التوقيت بعد المشارك المعني بها، كما هو معروض في الشكل رقم (٩-٥).



شكل رقم (٩-٥) تتم كتابة الحالات أفقياً في مخطط التوقيت بعد المشارك المرتبطة به.

٩-٥ الوقت Time

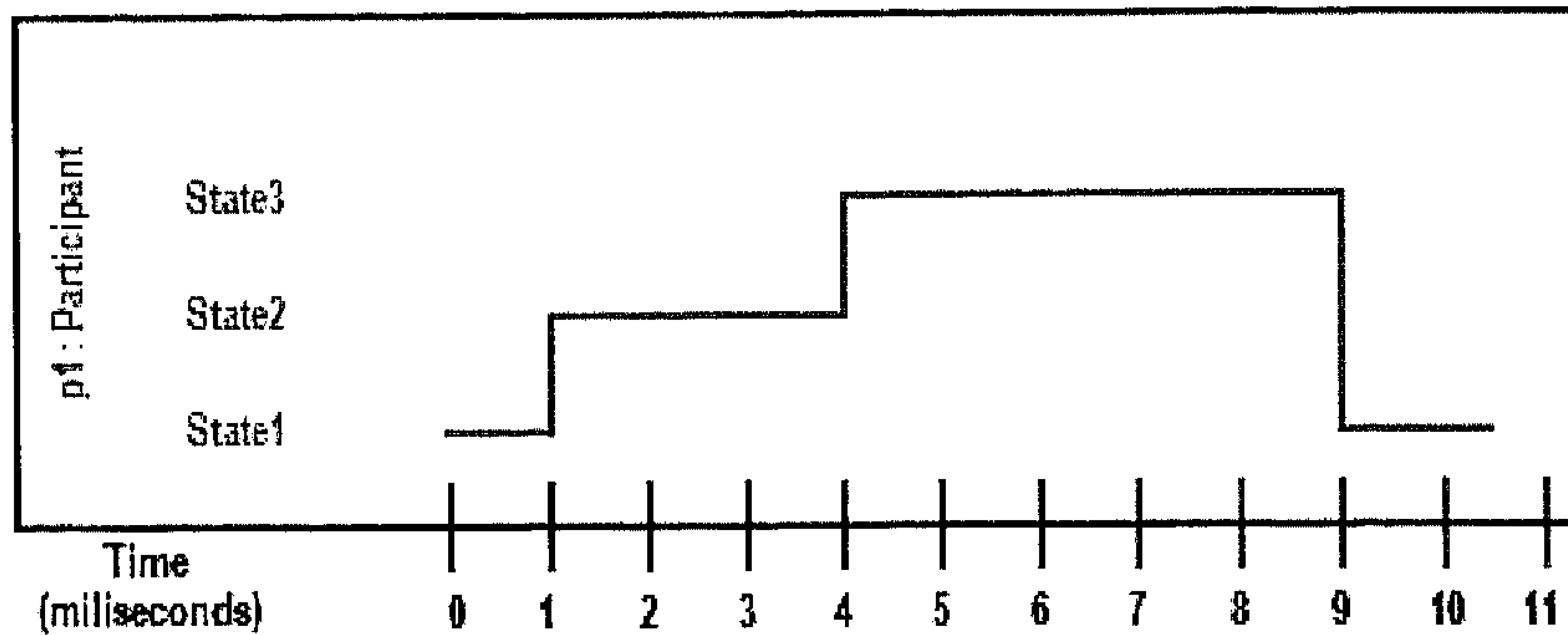
ربما يبدو غريباً بعض الشيء عدم ذكر الوقت بخصوص نوع مخطط يسمى حقاً مخطط "التوقيت". وقمنا حتى الآن بتهيئة المرحلة فقط، وذلك بإضافة المشاركين والحالات التي يمكن وضعهم على المخطط،

لكن حان الوقت لإضافة الوقت لنمذجة المعلومات المهمة فعلياً بالنسبة لمخطط توقيت.

٩-٥-١ مقاييس الوقت الدقيقة و مؤشرات الوقت النسبية

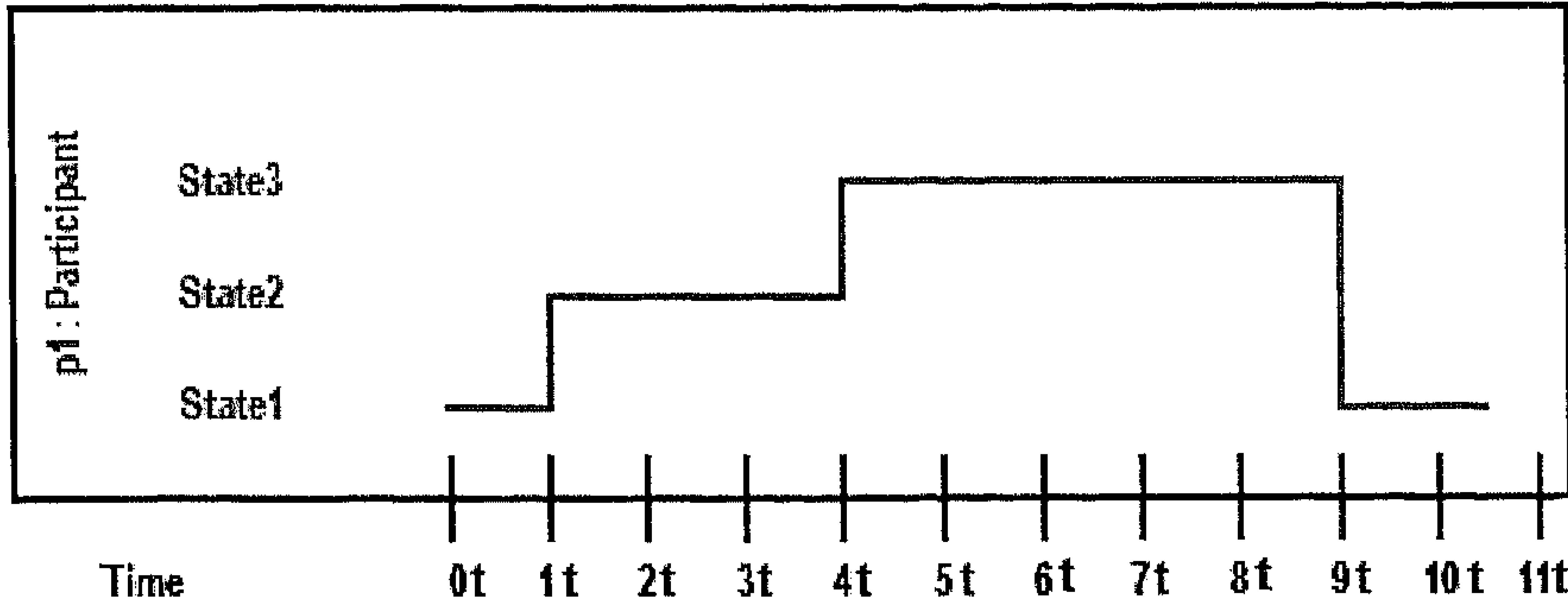
Exact Time Measurements and Relative Time Indicators

يتقدم الوقت على مخطط التوقيت من اليسار إلى اليمين عبر الصفحة، كما هو معروض في الشكل رقم (٩-٦).



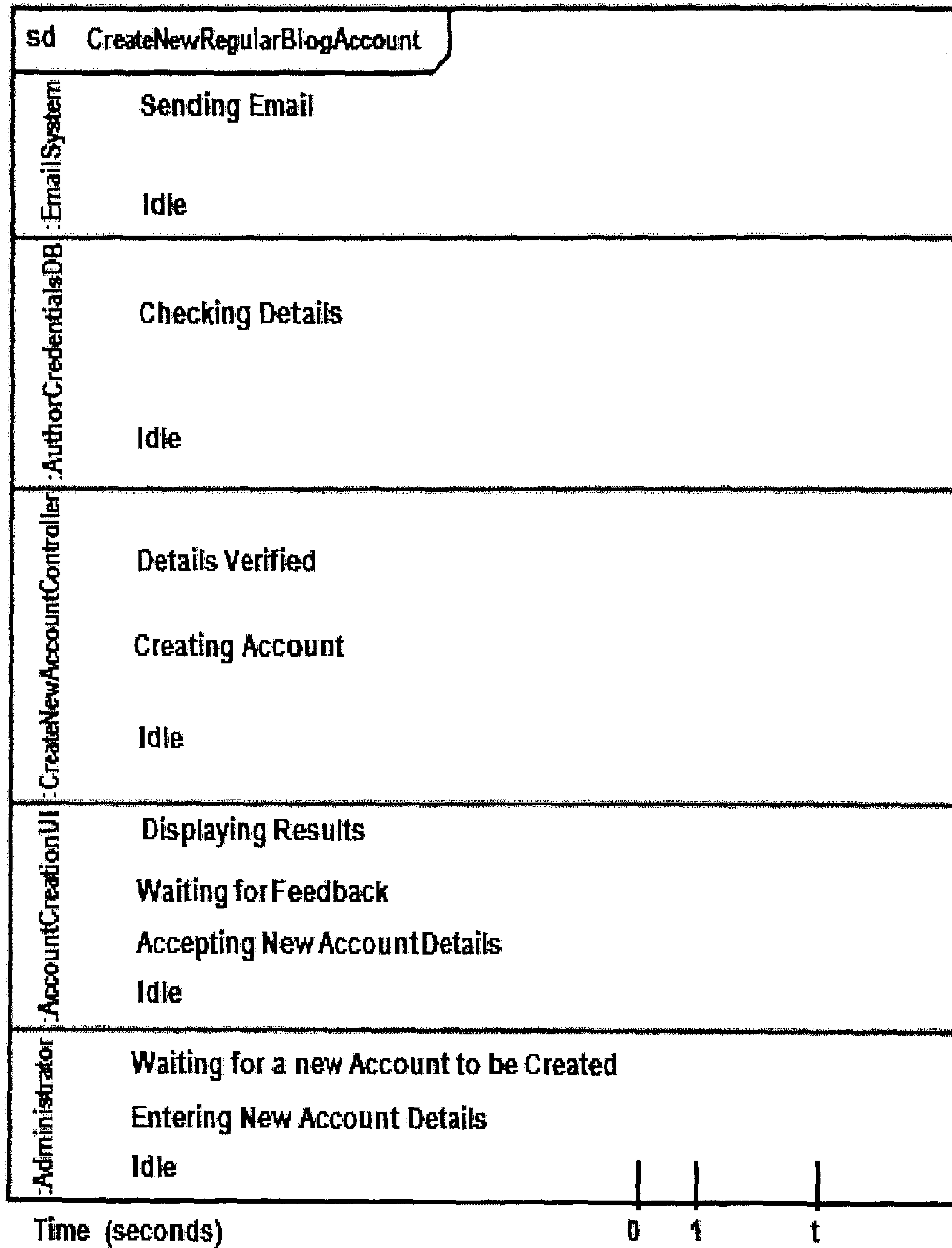
شكل رقم (٩-٦) تم وضع مقياس الوقت على مخطط التوقيت كمسطرة على طول أسفل الصفحة.

يمكن التعبير عن مقياس الوقت بعدة طرق مختلفة؛ يمكن أن يكون عندك مقياس وقت مضبوط، مثل الذي في الشكل رقم (٩-٦)، أو مؤشرات وقت نسبية، مثل الذي في الشكل رقم (٩-٧).



شكل رقم (٩-٧) تفيد مؤشرات الوقت النسبية كثيراً عندما يكون عندنا اعتبارات زمنية مثل "سيكون المشارك ParticipantA في الحالة State1 لمدة نصف الوقت الذي سيكون المشارك ParticipantB في الحالة State2".

تمثل t في مخطط التوقيت نقطة ذات أهمية في الزمن. لا تعرف بالضبط متى سيحدث الأمر؛ لأنه قد يحدث استجابة لرسالة أو حدث ما، لكن تشكل t وسيلة للإشارة إلى تلك اللحظة من دون معرفة متى ستكون بالضبط. وباستعمال t كمرجع، يمكن تحديد قيود الوقت بالنسبة لتلك النقطة t.



شكل رقم (٩-٨) إن قيود التوقيت هي مزيج من التواقيت المضبوطة والنسبية.

تعتبر إضافة الوقت إلى المخطط الذي قمنا بتجميعه إلى الآن أمراً معقداً بسبب عدم توفر أي معلومات توقيت عينية في المتطلب الأولي. انظر إلى القسم "قيود التوقيت في متطلبات النظام" سابقاً في هذا الفصل لتذكير سريع عن ما يشير إليه المتطلب الموسع أ-٢.

على أية حال، ما زلنا بحاجة إلى تطبيق القيود المذكورة في المتطلب أ-٢ ، لذلك تم عرض مقياس للوقت النسبي الممثل بالحرف t في الشكل رقم (٩-٨).

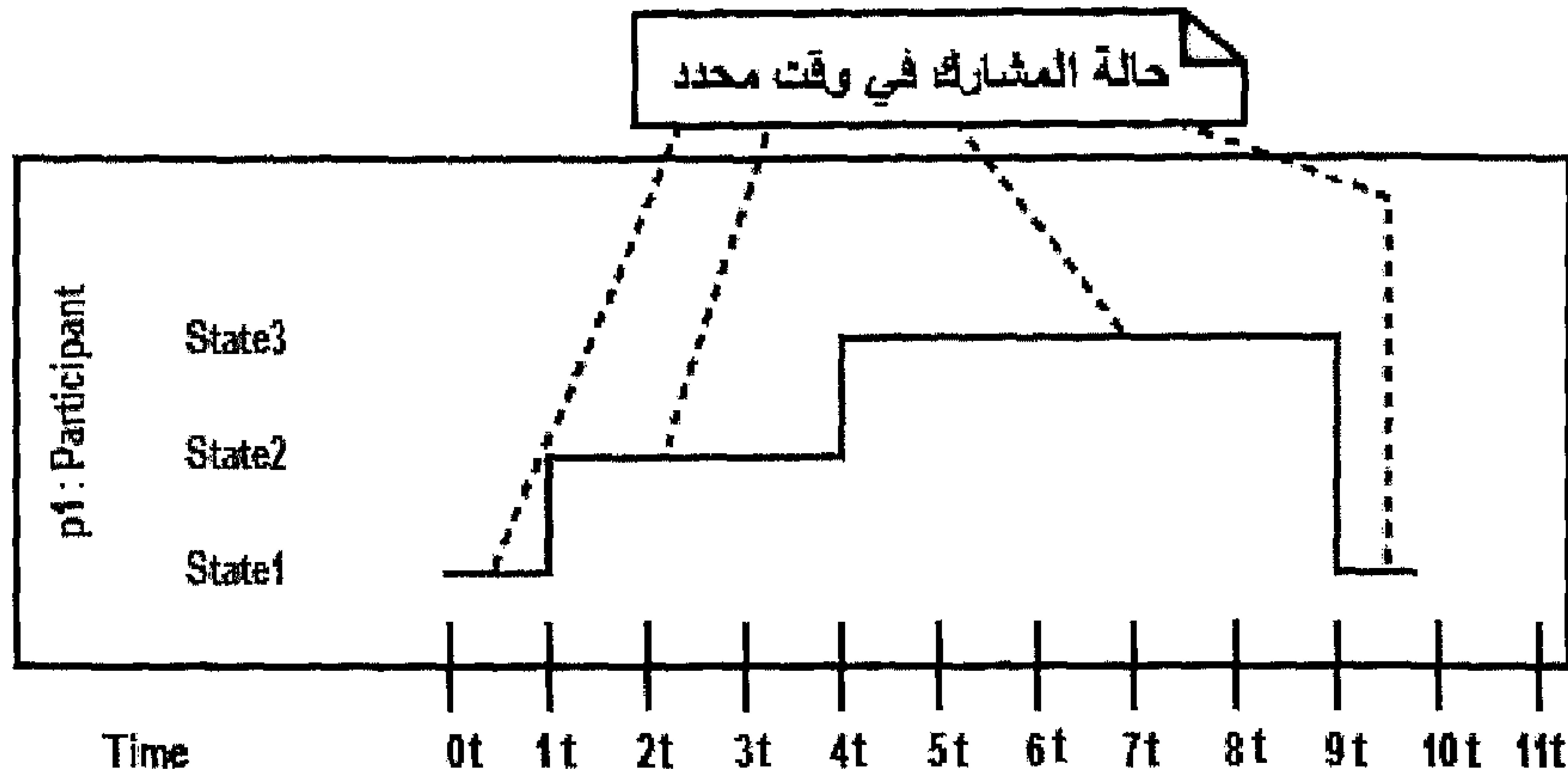
في الشكل رقم (٩-٨)، تم ببساطة قياس المراحل الأولية للتفاعل كثوان، وتمثل القيمة الفردية t ثانية فردية حيثما تكون مذكورة في أي قيود توقيت إضافية على المخطط. انظر إلى القسم "قيود التوقيت" لاحقاً في هذا الفصل للمزيد عن كيفية استعمال القيمة t على مخطط التوقيت.

٩-٦ خط حالة المشارك

A Participant's State-Line

بما أننا قمنا بإضافة الوقت إلى مخطط التوقيت، يمكن الآن عرض حالة المشارك في أي وقت محدد. وإذا نظرت للوراء إلى الشكل رقم (٩-٦) والشكل رقم (٩-٧)، يمكن ملاحظة كيف تم تحديد الحالة الحالية للمشارك من خلال استعمال خط أفقي يدعى خط الحالة.

ويكون خط حالة المشارك متراصفاً مع إحدى حالات المشارك في أي وقت أثناء التفاعل، انظر إلى الشكل رقم (٩-٩).

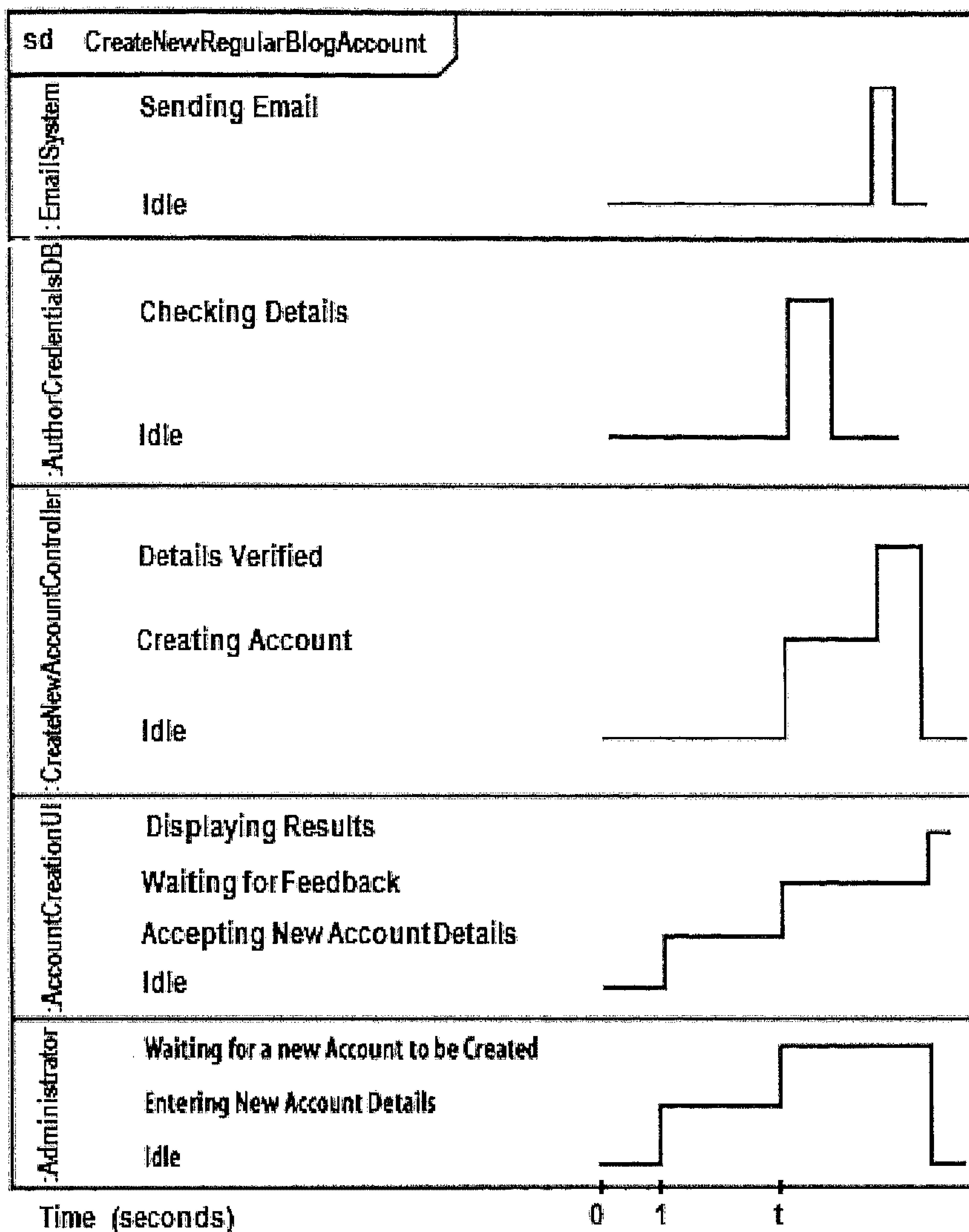


شكل رقم (٩-٩) يشير خط حالة المشارك p1:Participant إلى أنه في الحالة State1 لوحدة زمن واحدة، وفي الحالة State2 لثلاثة وحدات زمنية و في الحالة State3 لخمسة وحدات زمنية تقريباً (قبل العودة إلى الحالة State1 في نهاية التفاعل).

يعرض الشكل رقم (٩-١٠) كيفية تحديث مخطط توقيت إنشاء حساب مدونة عادي جديد، لعرض حالة كل مشارك في أي وقت أثناء التفاعل.

> في الحالات العملية، ربما تريد إضافة الأحداث و الحالات معاً إلى مخطط التوقيت وبنفس الوقت. لقد قمنا هنا ببساطة بتقسيم هذين النشاطين لتسهيل ملاحظة كيفية تطبيق هذين الجزأين من الترميز (من دون التباس الواحد مع الآخر).

كل ما يوجد للعرض هو أن المشارك موجود في حالة محددة بوقت معيّن. وقد حان الوقت الآن للنظر في المقام الأول إلى سبب تغيير حالة المشارك، والذي يقودنا بعناية إلى الأحداث و الرسائل.

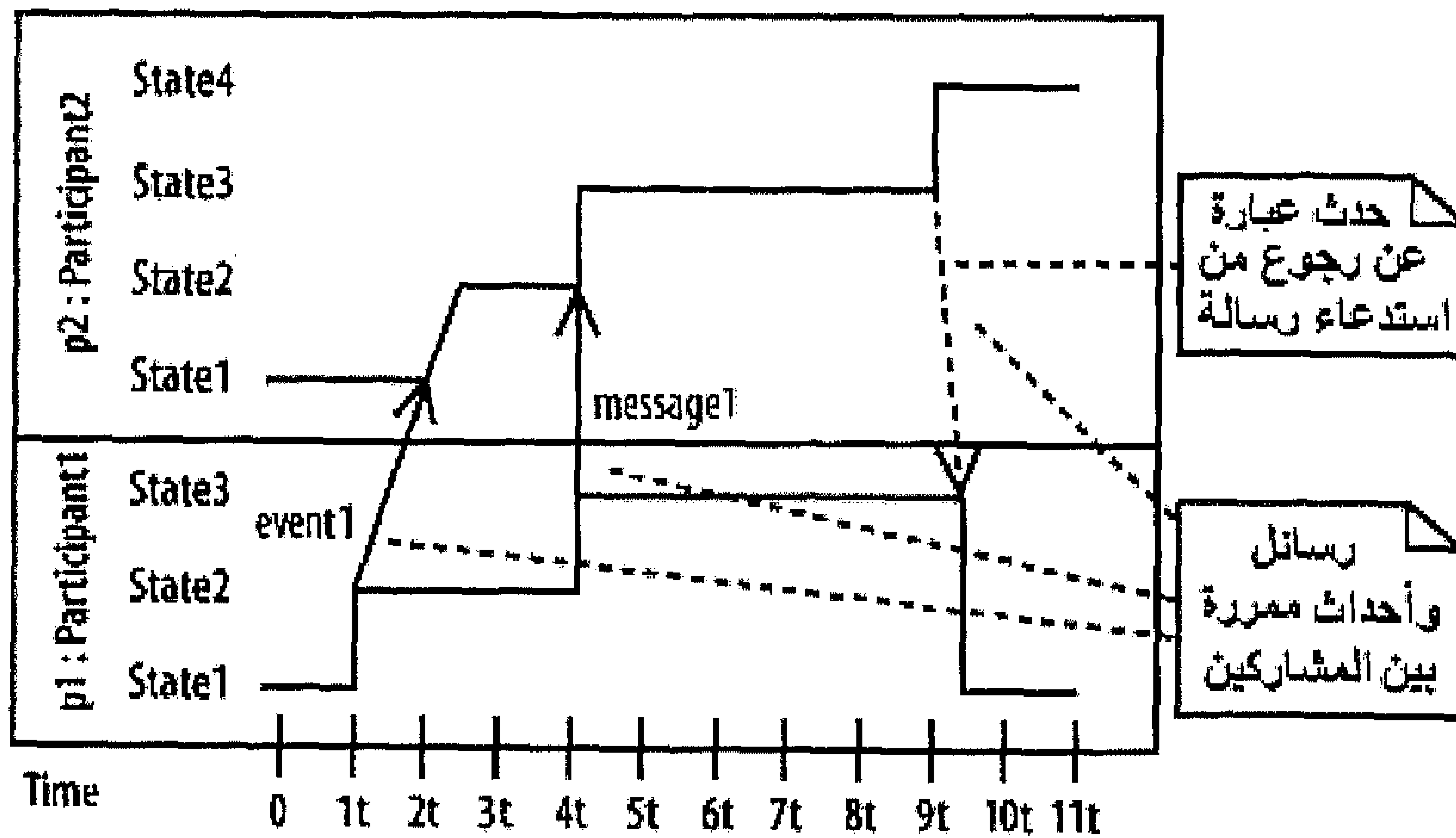


شكل رقم (٩-١٠) يحتاج كل مشارك إلى خط حالة موازٍ له لتحديد حالاته في أي نقطة من الزمن.

٩-٧ الأحداث و الرسائل
Events and Messages

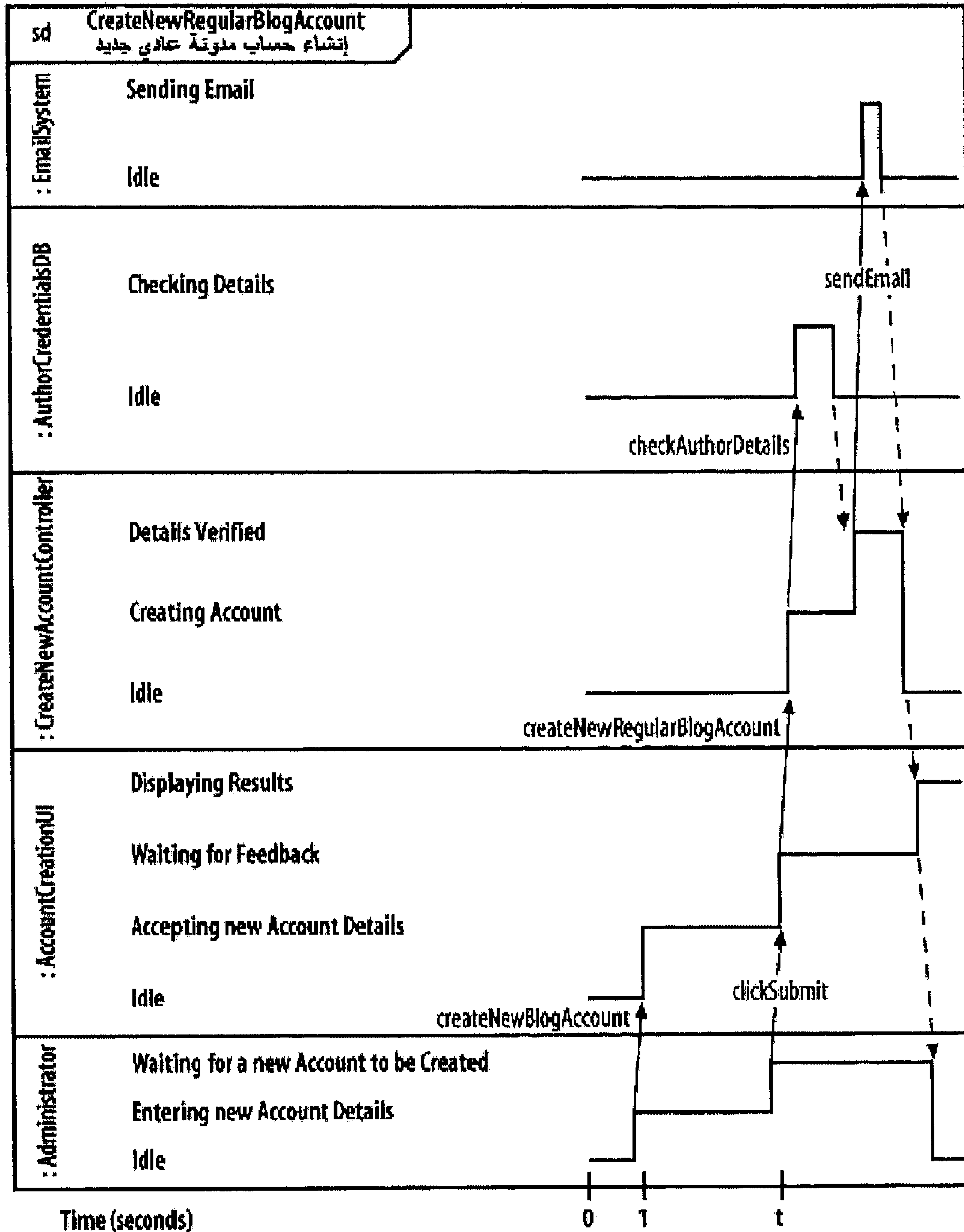
يقوم المشاركون بتغيير حالتهم في مخطط التوقيت استجابة للأحداث. قد تكون هذه الأحداث عبارة عن استدعاء رسالة ما أو ربما تكون شيئاً آخراً مثل الرجوع من رسالة بعد استدعائها. والاختلاف بين الرسائل والأحداث ليس مهما في مخطط التوقيت بنفس القدر الذي هو عليه في مخطط التتابع. إن الأمر المهم تذكره هو أنه مهما كان الحدث، فهو يعرض في مخطط التوقيت ليطلق تغييراً ما في حالة المشارك.

يتم عرض الحدث في مخطط التوقيت بواسطة سهم يبدأ من خط حالة المشارك (مصدر الحدث) إلى خط حالة مشارك آخر (مستلم الحدث)، كما هو معروض في الشكل رقم (٩-١١).



شكل رقم (٩-١١) يمكن أن يكون حتى للأحداث فترات زمنية خاصة بها في مخطط التوقيت، كما يظهر مع الحدث event1 الذي يأخذ وحدة زمن واحدة منذ استدعائه من قبل المشارك p1:Participant1 وحتى استلامه من قبل المشارك .p2:Participant2

أصبحت إضافة الأحداث إلى مخطط التوقيت عملاً بسيطاً تماماً وذلك من خلال الرجوع إلى مخطط التتابع في الشكل رقم (٩-٣) الذي يعرض الرسائل الممررة بين المشاركين، ويمكن إذن إضافة تلك الرسائل إلى مخطط التوقيت، كما هو معروض في الشكل (٩-١٢).

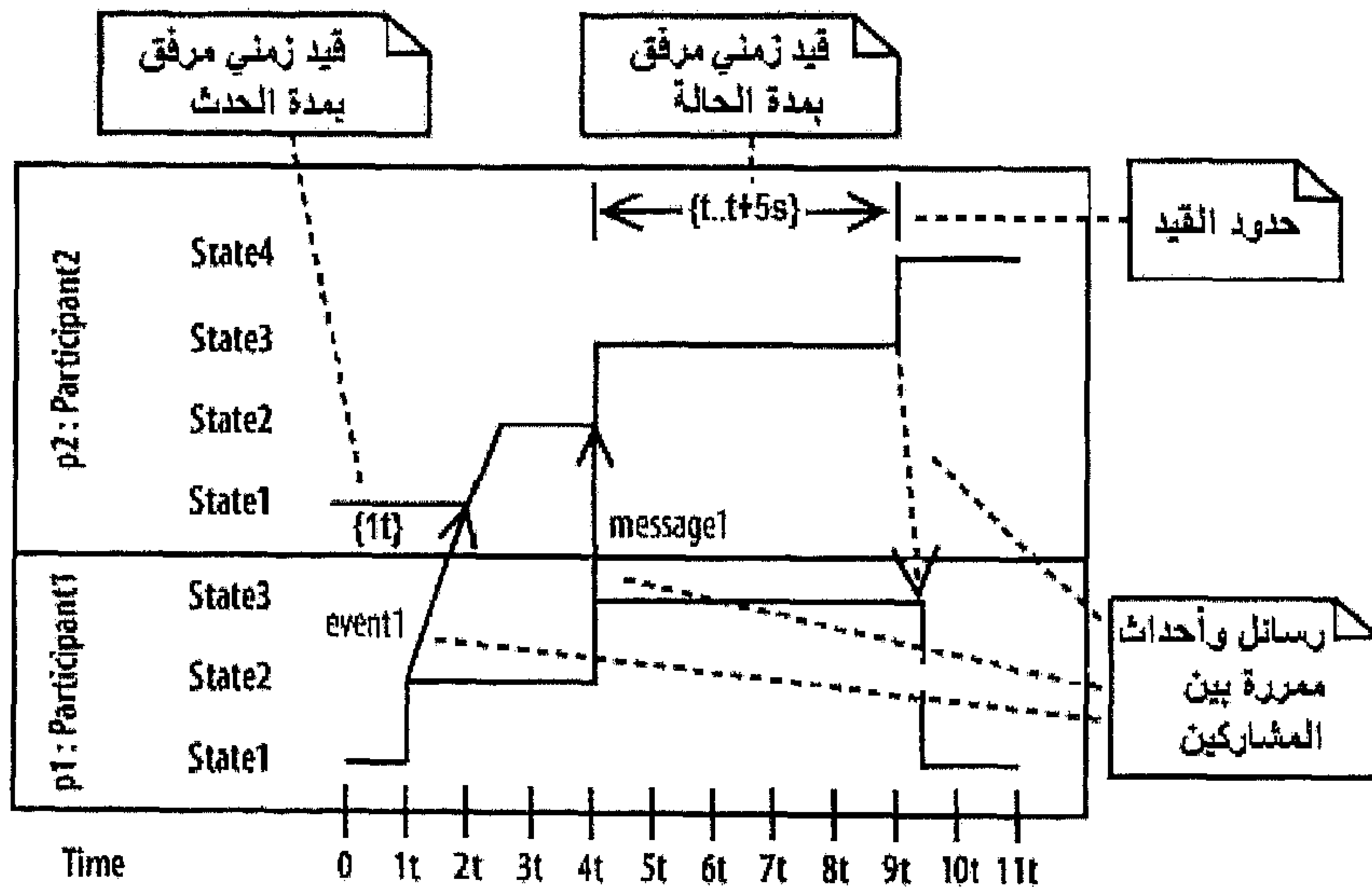


شكل رقم (٩-١٢) يتضح معنى تغييرات حالة المشارك أكثر بكثير عند رؤية الأحداث المسببة لها.

٩-٨ القيود الزمنية
Timing Constraints

حتى هذه النقطة، لقد قمنا حقاً بتحديد أساس مخطط التوقيت فقط. ويمثل المشاركون والحالات والوقت والأحداث والرسائل الخلفية الأمور التي يمكن الاستناد عليها للبدء بنمذجة المعلومات المهمة حقاً لمخطط التوقيت والمتمثلة بالقيود الزمنية.

وتصف القيود الزمنية بشكل مفصل الوقت الذي سيأخذه جزء محدد من التفاعل. عادة ما يتم تطبيق هذه القيود على المدة الزمنية التي يكون المشارك فيها بحالة محددة، أو الوقت الذي سيأخذه حدث ما ليتم استدعاؤه واستلامه، كما هو معروض في الشكل رقم (٩-١٣).



شكل رقم (٩-١٣) قد ترفق القيود الزمنية بحدث أو حالة ما و بأسهم حدود القيد.

بتطبيق القيود الزمنية على مخطط التوقيت في الشكل رقم (٩-١٣)، يمكننا القول أنه يجب أن تكون مدة الحدث event1 أقل من وحدة قياس

نسبي t واحدة، و أن يبقى المشارك p2:Participant2 في الحالة State4 خمسة ثوان على الأكثر.

٩-٨-١ بنية القيد الزمني Timing Constraint Formats

يمكن بناء القيد الزمني بعدة طرق مختلفة، وذلك بالاعتماد على المعلومات التي نحاول نمذجتها. ويعرض الجدول رقم (٩-١) بعض الأمثلة الشائعة للقيود الزمنية.

جدول رقم (٩- ١) الطرق المختلفة لتحديد قيد زمني.

القيد الزمني	الوصف
{t..t+5s}	يجب أن تكون مدة الحدث أو الحالة خمسة ثوان أو أقل.
{<5s}	يجب أن تكون مدة الحدث أو الحالة أقل من خمسة ثوان. هذه الصيغة مألوفة أقل بقليل من الصيغة {t..t+5s} ولكنها توازيها.
{>5s, <10s}	يجب أن تكون مدة الحدث أو الحالة أكبر من خمسة ثوان ولكن أقل عشرة ثوان.
{t}	يجب أن تكون مدة الحدث أو الحالة تساوي القيمة t، هذا مقياس نسبي حيث يمكن أن تكون t أية قيمة زمنية.
{t..t*5}	يجب أن تكون مدة الحدث أو الحالة مضاعف قيمة t خمس مرات، هذا مقياس نسبي آخر (يمكن أن تكون t أية قيمة زمنية).

٩-٨-٢ تطبيق القيود الزمنية على الحالات و الأحداث
Applying Timing Constraints to States and Events

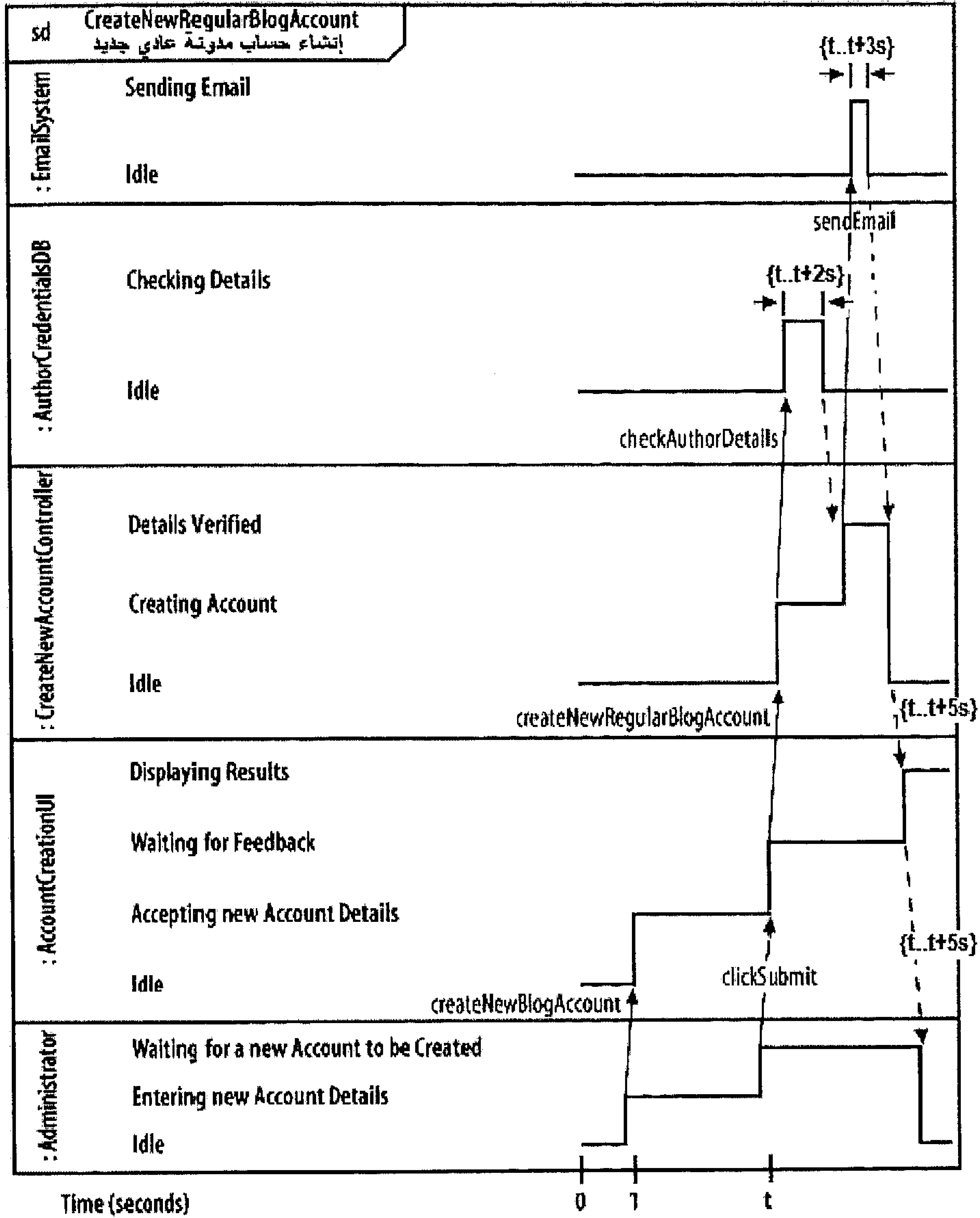
في بداية هذا الفصل، قمنا بتوسعة المتطلب أ-٢ من خلال تحديد بعض الاعتبارات الزمنية فيه. ويمكن الآن إضافة اعتباراته الزمنية إلى مخطط التوقيت على شكل قيود زمنية. ويقوم الشكل رقم (٩-١٤) بتكملة مخطط توقيت إنشاء حساب مدونة عادي جديد من خلال أسر اعتبارات المتطلب أ-٢ الزمنية بتطبيق القيود الزمنية على الحالات ذات العلاقة.

كما يمكنك ملاحظته من الشكل رقم (٩-١٤)، فإن تطبيق القيد الزمني "استغراق إنشاء حساب مدونة عادي جديد خمسة ثوان" ليس عملاً بسيطاً، بسبب تأثيره في عدة تفاعلات متداخلة مختلفة بين المشاركين. وتأتي هنا مهارة المنمذج لتؤدي دوراً في مخطط التوقيت؛ عليك أن تقرر أي أحداث أو حالات تحتاج أن يخصص لها أجزاء من الخمسة ثوان المتوفرة، كي يتمكن كل مشارك من القيام بعمله (والحصول على تلك التخصيصات الزمنية بشكل سليم).

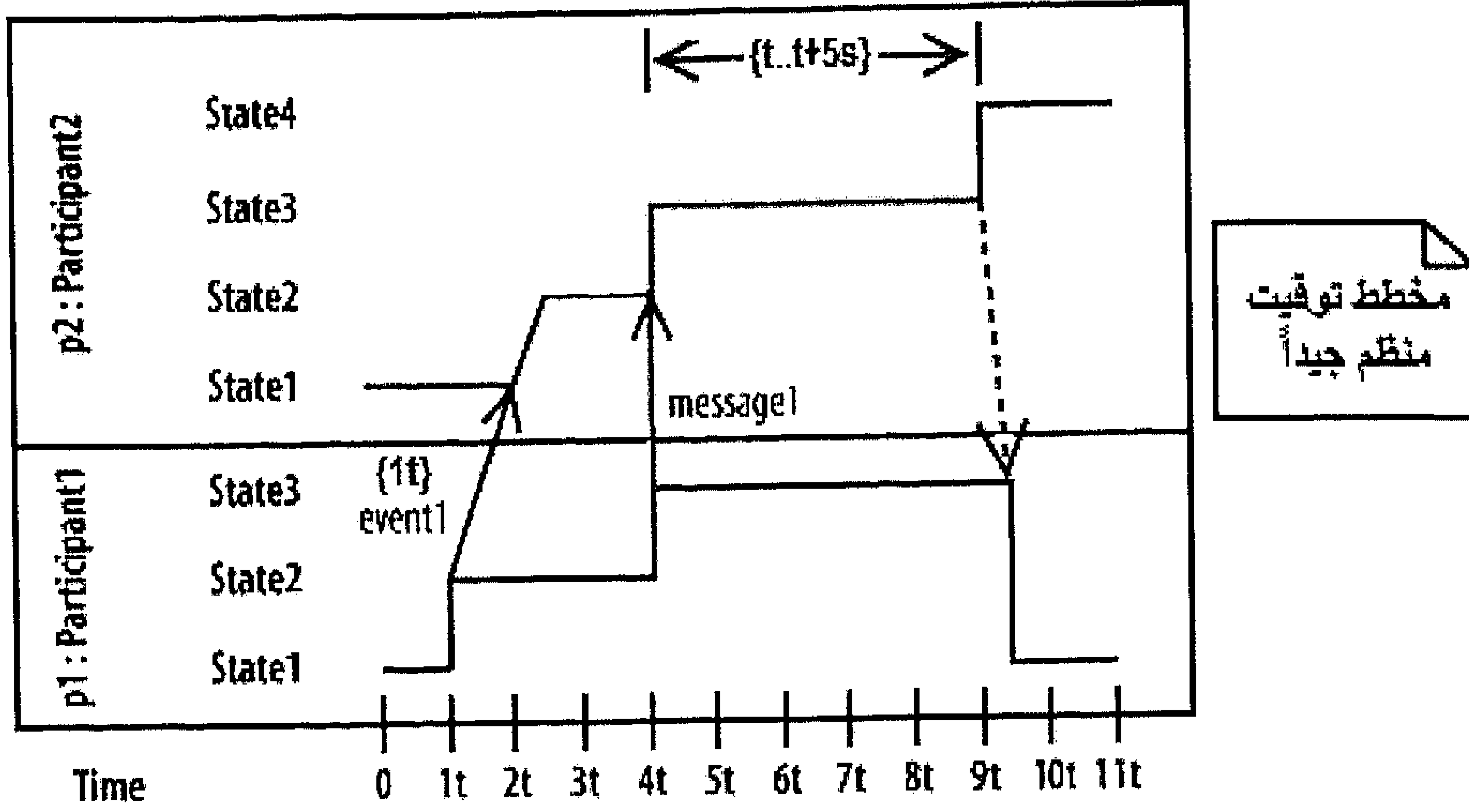
٩-٩ تنظيم المشاركين على مخطط التوقيت

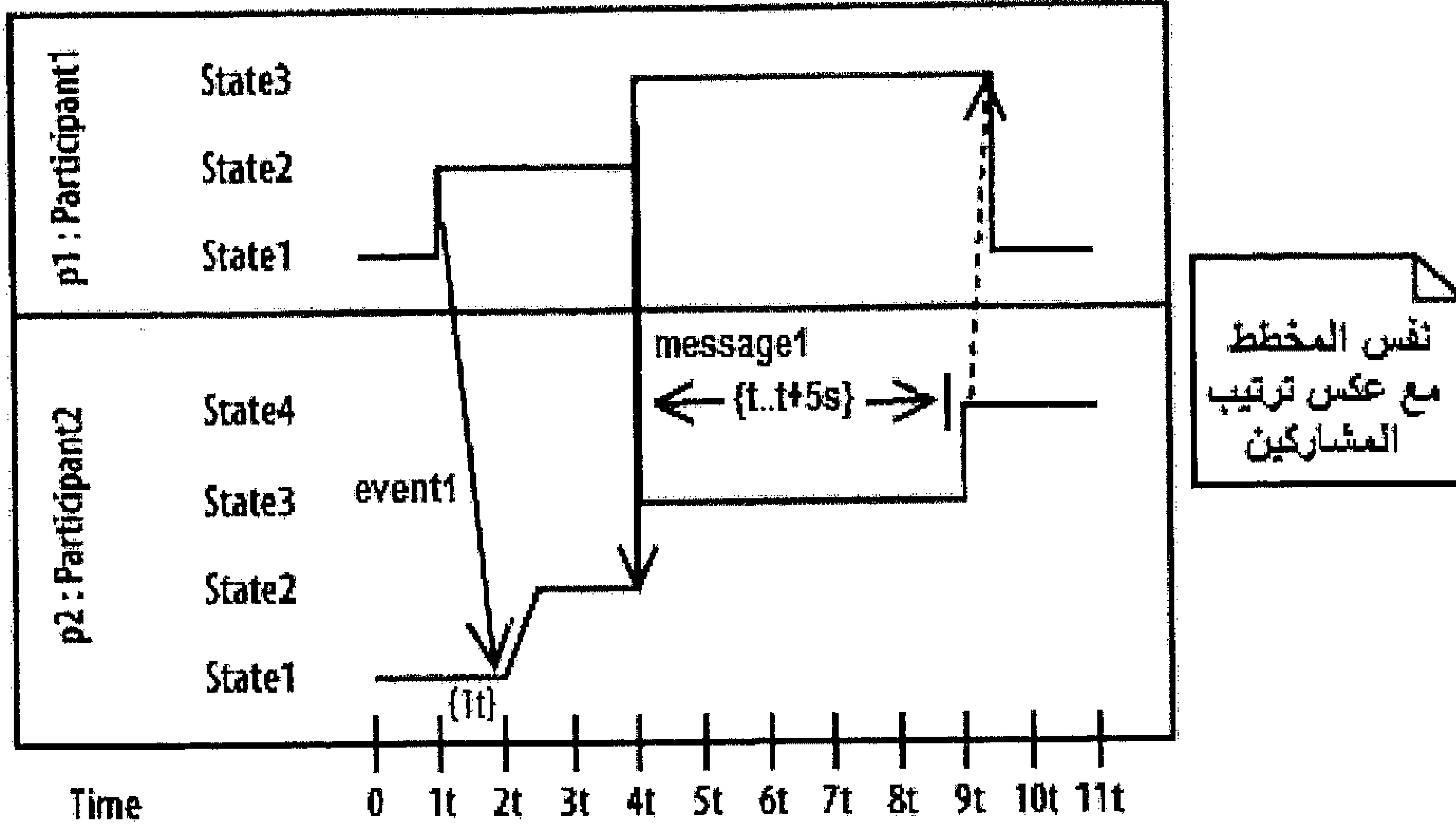
Organizing Participants on a Timing Diagram

لا يهم كثيراً المكان الذي تضع فيه المشاركين بالبداية على مخطط التوقيت. على أية حال، بينما تضيف تفاصيل أكثر على شكل أحداث ومعلومات زمنية إلى المخطط، ستكتشف سريعاً أن المكان الذي وضعت المشارك فيه على مخطط التوقيت قد يسبب المشاكل إذا لم تفكر بعناية كافية بخصوصه، انظر الشكل رقم (٩-١٥).



شكل رقم (٩-١٤) لم يمضِ أكثر من خمسة ثوان منذ نقر المشارك **Administrator**: على الزر أرسل **submit** حتى النقطة التي أنشأ النظام عندها الحساب الجديد.

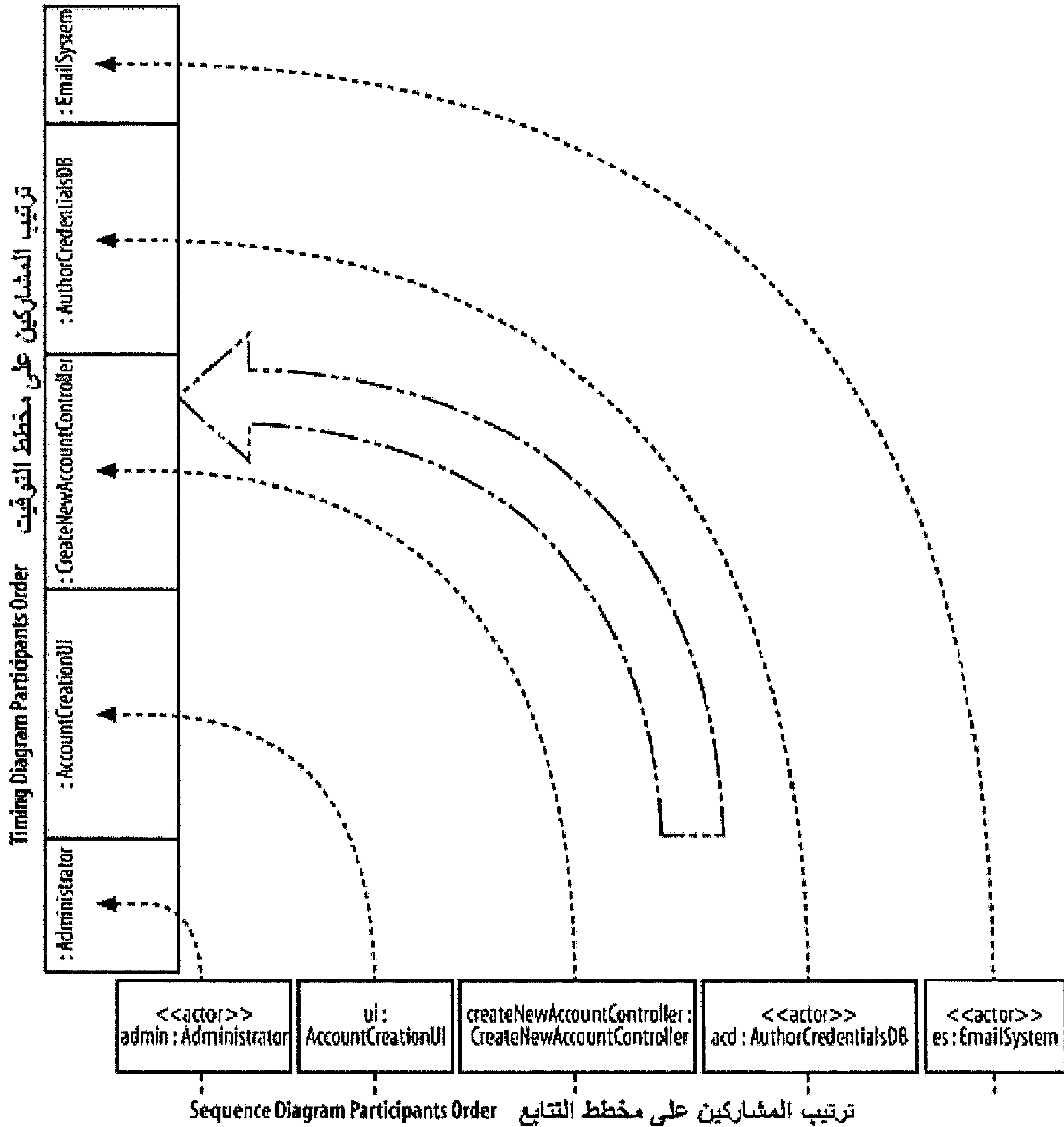




شكل رقم (٩-١٥) إن المخطط السفلي أصعب قراءة حيث بدأت التفاصيل تحجب بعضها بعضا.

إذا كنت محظوظاً و كان يتوفر لديك مخطط تتابع للتفاعل، فيوجد طريقة مجربة سهلة للبدء بترتيب المشاركين للمرة الأولى على مخطط التوقيت. خذ ببساطة ترتيب المشاركين حسب ظهورهم على طول أعلى الصفحة في مخطط التتابع، واقلب قائمة أسماء المشاركين ٩٠ درجة

بعكس عقارب الساعة، كما هو معروض في الشكل رقم (٩-١٦). إذا كان مخطط التتابع منظماً بشكل جيد، فيجب أن تحصل الآن على مرشح جيد لترتيب وضع المشاركين على مخطط التوقيت.



شكل رقم (٩-١٦) يشكل دوران المشاركين الرئيسيين في مخطط التتابع ٩٠ درجة بعكس عقارب الساعة طريقة سهلة للحصول على موقع أولي لمشاركي مخطط التوقيت.

٩-١٠ الترميز البديل
An Alternate Notation

يشكل امتلاك لغة النمذجة الموحدة عديداً من المخططات دليل عافية دائم. وفي النسخ السابقة من لغة النمذجة الموحدة، كانت مخططات التتابع معروفة بأنها الأكثر صعوبة في إدارتها عند نمذجة تفاعل معقد. ورغم أن مخطط التوقيت لن يسرق إنجاز مخطط التتابع في هذا المجال، فإن الترميز العادي لمخطط التوقيت، المذكور حتى الآن في هذا الفصل، لا يناسب بشكل جيد الحاجة إلى نمذجة عدد كبير من الحالات المختلفة.



لقد تم تخفيف بعض المشاكل مع مخططات التتابع بإدراج أقسام التتابع في UML 2.0 (انظر إلى الفصل السابع).

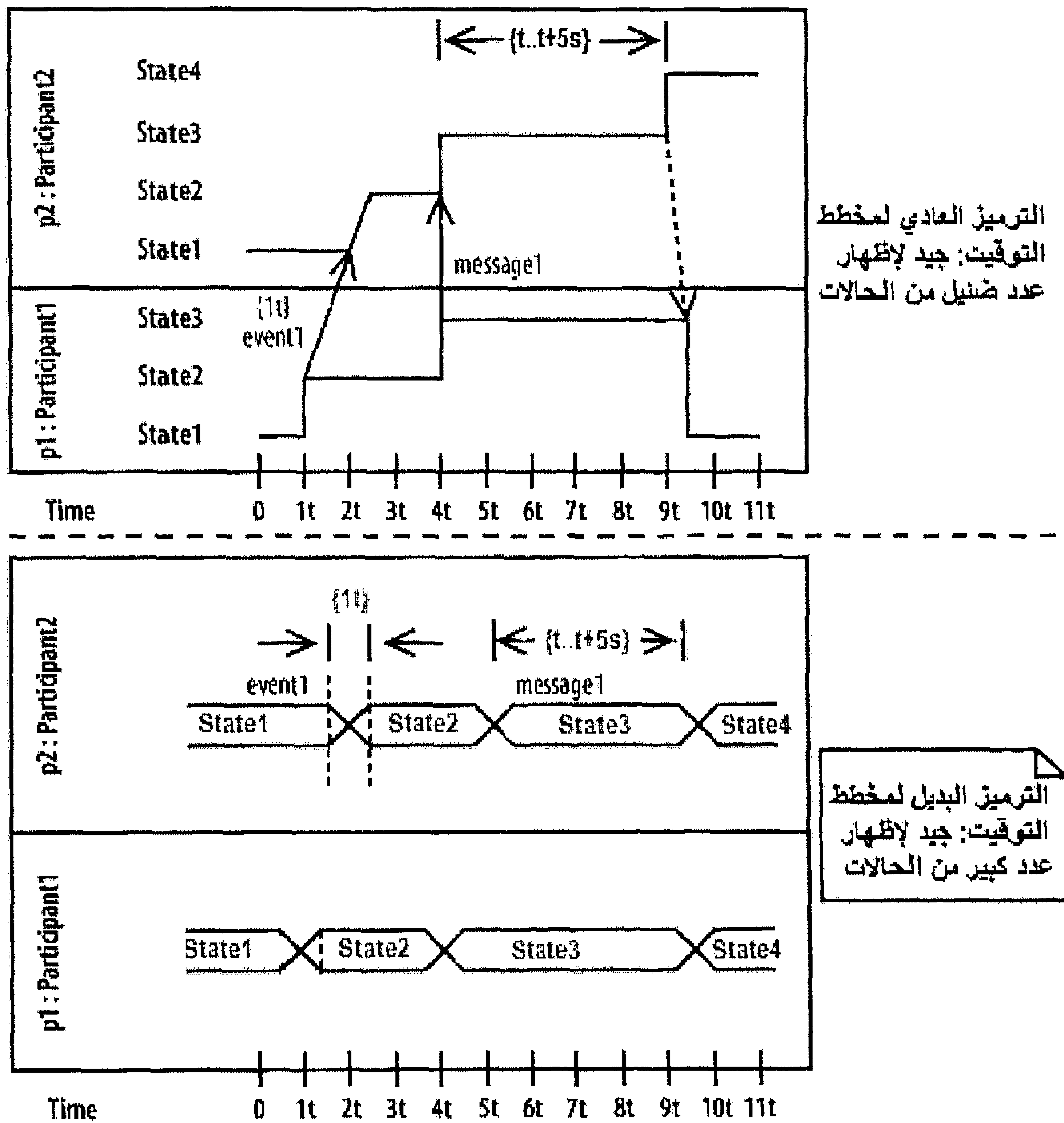
ويعتبر مخطط توقيت التفاعل "إنشاء حساب مدونة عـادي جديد" – كما هو معروض في الشكل رقم (٩-١٤) – مثالاً بسيطاً جداً. على أية حال، لعلك بدأت تستوعب كم سيكبر (على الأقل بشكل عمودي) مخطط توقيت أي تفاعل أكبر من تفاعل بديهي يتضمن عدداً ضئيلاً من الحالات. وقد تساعدك أداة جيدة للغة النمذجة الموحدة بالعمل معها وإدارة مخططات التوقيت الكبيرة، لكن هناك محدودية لما يمكن أن تَعمله حقاً هكذا أداة.

أدرك مطورو UML 2.0 هذه المشكلة، فقاموا بإنشاء ترميز بديل سهل الاستعمال للتفاعلات المحتوية على عدد كبير من الحالات، انظر في الشكل رقم (٩-١٧).

عند النظر بعناية في الترميز البديل لمخطط التوقيت، ترى أن الأشياء لم تختلف بشكل مثير عن الترميز العادي. ولم يتغير ترميز

المشاركين والوقت إطلاقاً، إنما التغيير الأبرز بين الترميز العادي والترميز البديل لمخطط التوقيت هو في كيفية عرض الحالات وتغييرات الحالة.

ويعرض الترميز العادي لمخطط التوقيت الحالات كقائمة بجانب المشارك المعني. وهناك حاجة بالتالي إلى خط حالة لعرض بأي حالة يكون المشارك في وقت محدد. لسوء الحظ، إذا كان للمشارك العديد من الحالات المختلفة، فسيكبر بشكل سريع حجم المكان الضروري لنمذجة المشارك في مخطط التوقيت.



شكل رقم (٩-١٧) يجب أن يكون ترميز المخطط العلوي مألوفاً لك، ولكن يستعمل المخطط السفلي ترميز مخطط التوقيت البديل الجديد.

ويعالج الترميز البديل هذه المشكلة بإزالة القائمة العمودية للحالات المختلفة. ويقوم بوضع حالات المشارك مباشرة عند النقطة الزمنية التي يكون المشارك عندها في تلك الحالات. لذلك، لم يعد هناك حاجة لخط الحالة، ويمكن وضع كل حالات المشارك في خط حالة فريد على طول المخطط.

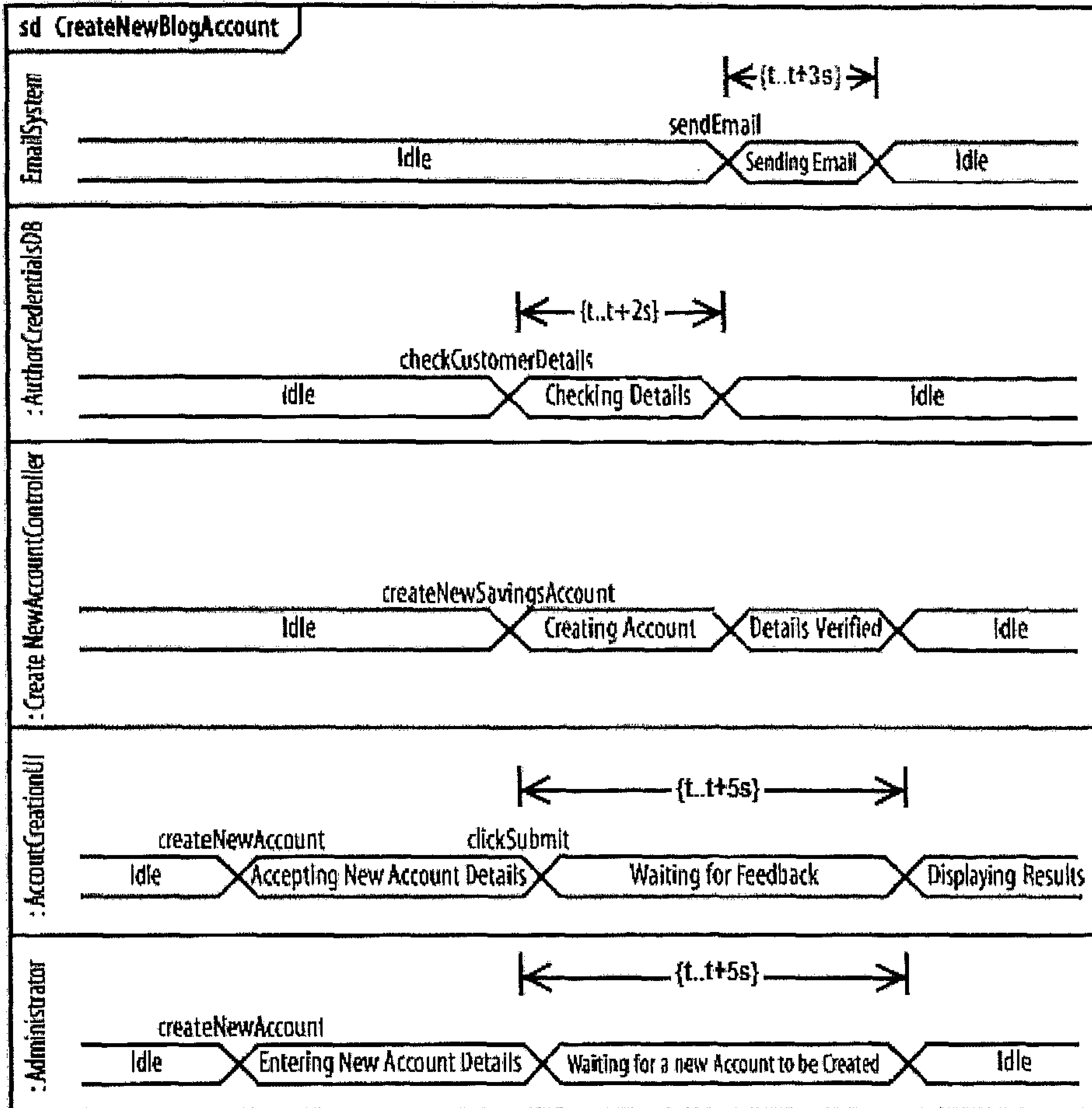
ولإظهار تغيير حالة المشارك بسبب حدث ما، يتم وضع الإشارة (X) بين الحالتين، وتتم كتابة الحدث المسبب لتغيير الحالة بجانب الإشارة X. يمكن بالتالي تطبيق القيود الزمنية بنفس الطريقة مع الترميز العادي.

وللوصول بموضوع مخططات التوقيت إلى نهاية ما، يعرض الشكل رقم (٩-١٨) ترميز مخطط التوقيت البديل في إطار عملي من خلال نمذجة التفاعل "إنشاء حساب مدونة عادي جديد".

ترميز آخر لمخططات التوقيت

A Second Notation for Timing Diagrams

لماذا يكون عندنا ترميز آخر لمخططات التوقيت؟ لحسن الحظ، هناك جواب سهل لهذا السؤال: فإن الترميز العادي لمخطط لا يناسب بالقدر المطلوب التوقيت بكل بساطة، وذلك عند تواجد العديد من المشاركين الذين قد يأخذون حالات عديدة مختلفة أثناء حياة التفاعل. كما أن الجواب كان سهلاً جداً، كذلك كانت الطريقة المجربة التي يمكن استعمالها للمساعدة في اختيار الترميز الذي تتبناه لتفاعل محدد. في حال تم وضع المشارك في العديد من الحالات المختلفة أثناء مسار حياة التفاعل، ويستحق حينئذ الأخذ بالاعتبار استعمال الترميز البديل. وفي الحالات الأخرى، قم باستعمال الترميز العادي لأنه معترف به بشكل أوسع في مجتمع النمذجة.



شكل رقم (٩-١٨) بالرغم من أنه ليس هناك العديد من الحالات في هذا التفاعل، فيمكنك البدء برؤية كيف أن الترميز البديل هو أكثر تراصاً و أسهل إدارة، في حالة وجود العديد من الحالات لكل مشارك.

٩-١١ ما هي الخطوة التالية؟

يتكامل مفهوم حالة الكائن مع مخططات التوقيت، لأنه يعرض حالات الكائن في أوقات محددة. يعرض مخطط حالة الآلة بشكل مفصل حالات الكائن والمُطلقات المسببة لتغييرات الحالة. لهذَين النوعَين من المخططات أهمية كبيرة في نمذجة الأنظمة الفورية والأنظمة الضمنية. ستتم تغطية مخططات حالة الآلة في الفصل الرابع عشر.

إتمام وصف التفاعل: مخططات ملخّص التفاعل

COMPLETING THE INTERACTION PICTURE: INTERACTION OVERVIEW DIAGRAMS

لم يكن مطلوباً إطلاقاً "النظر إلى الوصف العام"؟ سواء كنت تعمل على فكرة جديدة أو تنمذج في لغة النمذجة الموحدة، قد يساعد أحياناً الرجوع من التفاصيل للشعور بشكل أفضل تجاه ما عملته والسياق الذي عملته فيه. هذا هو عمل مخططات ملخّص التفاعل؛ وهي موجودة لتوفير منظور الوصف العام لتفاعلات النظام.

توفر مخططات ملخّص التفاعل منظوراً عالي المستوى عن كيفية عمل عدة تفاعلات معاً لإنجاز مهام النظام، مثل حالة الاستخدام. وتركز مخططات التتابع ومخططات الاتصال ومخططات التوقيت على تفاصيل محددة متعلقة بالرسائل المؤلفة للتفاعل، لكن تربط مخططات ملخّص التفاعل التفاعلات المختلفة معاً في وصف وحيد وكامل للتفاعلات المؤلفة للأمور المهمة في النظام.

ويشبه ملخّص التفاعل بشكل كبير مخطط النشاط (انظر إلى الفصل الثالث)، باستثناء أن كل الأفعال هي تفاعلات بحد ذاتها. فكر

بكل جزء من ملخص التفاعل كأنه تفاعل كامل بحد ذاته. وإذا كان تفاعل ما ضمن الملخّص أكثر ارتباطاً بالتوقيت، فتستطيع حينئذ استخدام مخطط التوقيت (انظر إلى الفصل التاسع)، وربما يحتاج تفاعل آخر داخل الملخّص إلى التركيز على ترتيب الرسائل، فتستطيع حينئذ استعمال مخطط التتابع (انظر الفصل السابع). ويقوم ملخّص التفاعل بربط التفاعلات المنفصلة داخل النظام معاً في الترميز ليعطي معنى أكثر لتفاعل محدد، وذلك لعرض كيفية عمل التفاعلات معاً لتنفيذ الأمور المهمة في النظام.

١٠-١ أجزاء مخطط ملخّص التفاعل

The Parts of an Interaction Overview Diagram

أفضل طريقة لفهم ترميز مخطط ملخّص التفاعل هي بالتفكير فيه كمخطط نشاط، باستثناء أنه بدلاً من توصيف الفعل، فيتم توصيف كامل التفاعل باستعمال المخطط الخاص به، كما في الشكل رقم (١٠-١).

يمكن أن يشترك أي عدد من المشاركين في التفاعلات التي تحدث ضمن الملخّص. لرؤية المشاركين مشتركين خلال كامل الملخّص، يتم إضافة عنوان فرعي لخطوط الحياة إلى عنوان المخطط، كما هو معروض في الشكل رقم (١٠-٢)..

بشكل مشابه لمخطط النشاط، يبدأ ملخّص التفاعل بعقدة بداية و ينتهي بعقدة نهاية. ويتدفق التحكم بين هاتين العقدتين حيث يمر عبر كل التفاعلات التي بينهما. وعلى أية حال، لست مقيّداً بتدفق تتابعي بالذات بين التفاعلات.

iod : InteractionOverview1

sd Interaction1

mCaller: MessageCaller | mReceiver:MessageReceiver | mReceiver2 :MessageReceiver2

foo()

bar()

bar()

cd Interaction2

participant:1: ParticipantClassA — 1.messageA() — participant:2: ParticipantClassB

2 messageB()

cd Interaction3

participant:1: ParticipantClassA — 1.messageA() — participant:2: ParticipantClassB

2 messageB()

cd Interaction4

participant:1: ParticipantClassA — 1.messageA() — participant:2: ParticipantClassB

2 messageB()

sd Interaction5

mCaller: MessageCaller | mReceiver:MessageReceiver | mReceiver2 :MessageReceiver2

foo()

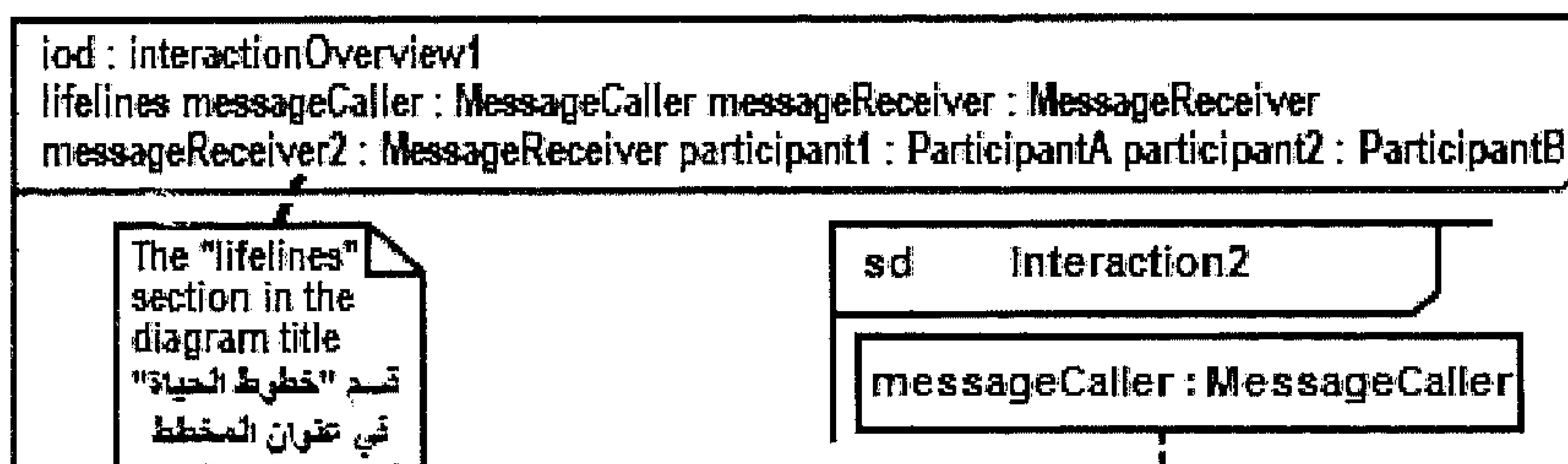
bar()

bar()

Each "Action" is an interaction in its own right

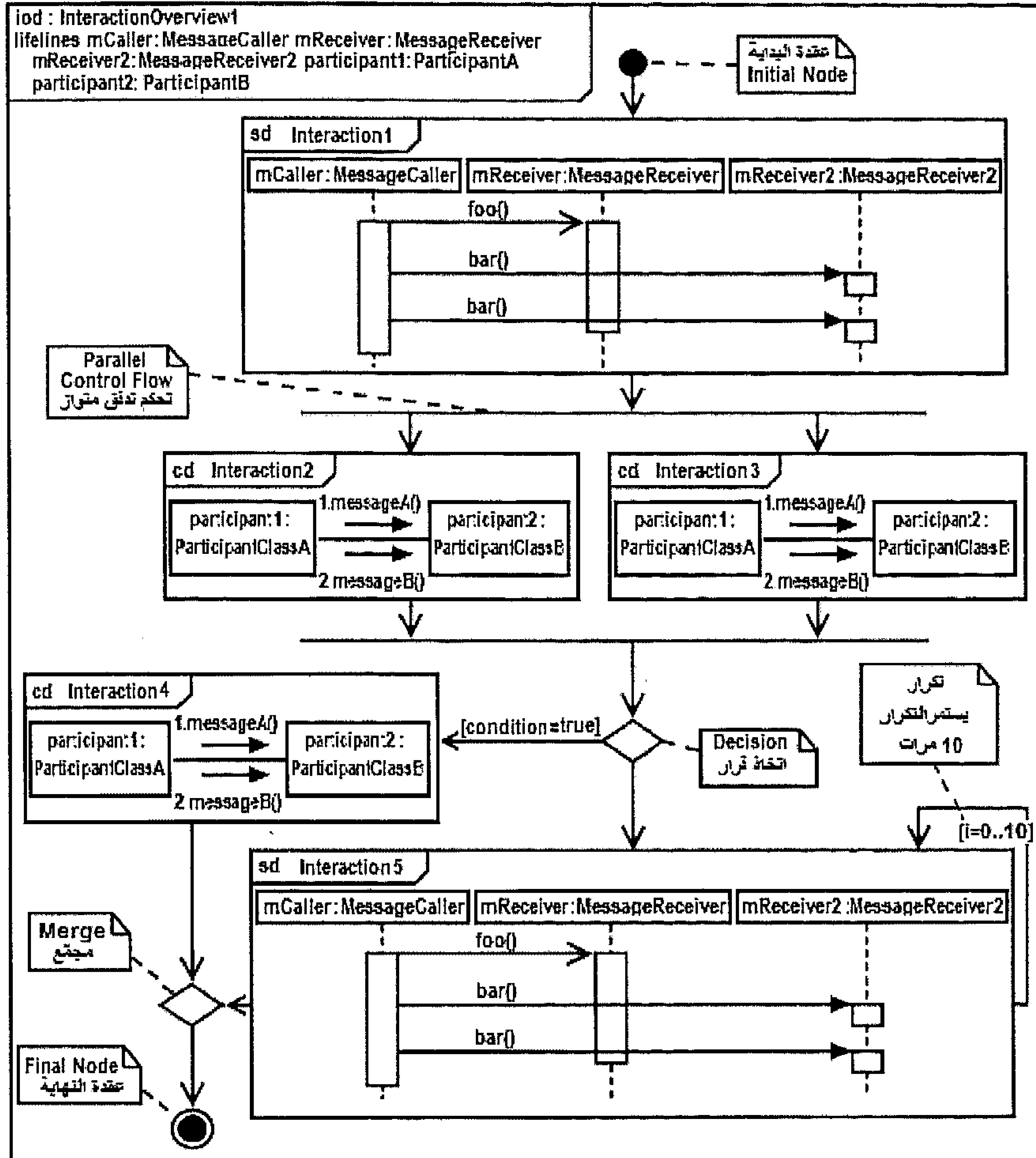
كل "فعل" هو تفاعل بحد ذاته

شكل رقم (١٠-١) تم وضع التفاعلات الفردية على مخطط ملخّص التفاعل كأنها أفعال على مخطط النشاط.



شكل رقم (١٠-٢) يعرض العنوان الفرعي لخطوط الحياة القائمة المركبة من المشاركين المشتركين في التفاعلات داخل الملخّص.

كما أنه يمكن لتدفق التحكم في مخطط النشاط أن يخضع إلى القرارات و الأعمال المتوازية و حتى التكرارات، فيمكن أن يتم ذلك أيضاً في مخطط التفاعل، كما هو معروض في الشكل رقم (١٠-٣).



شكل رقم (١٠-٣) نبدأ بعقدة البداية، ثم يتم تنفيذ التفاعل **Interaction1** متبعاً بشكل متوازي بتنفيذ التفاعلَين **Interaction2** و **Interaction3**؛ وينفذ التفاعل **Interaction4** إذا كان الشرط **condition=true** صحيحاً؛ و إلا فيتم تنفيذ التفاعل **Interaction5** عشر مرات في حلقة قبل دمج تدفق التحكم و الوصول إلى عقدة النهاية.

١٠-٢ نمذجة حالة استخدام باستعمال ملخص التفاعل
Modeling a Use Case Using an Interaction Overview

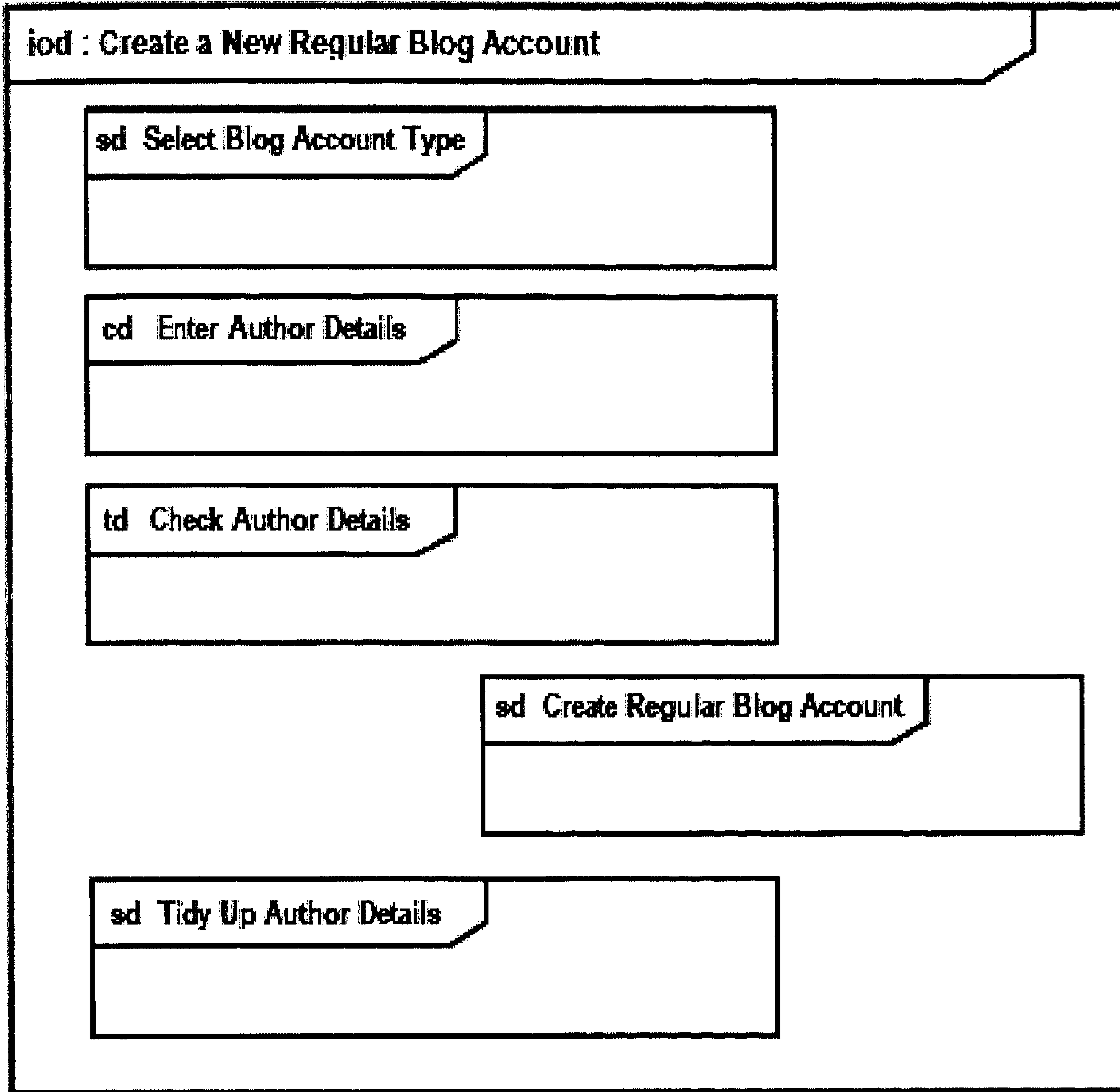
هذا كل ما يخص الترميز الجديد لمخططات التفاعل؛ حان الوقت الآن للنظر في مثال عملي. لتهيئة المرحلة، وسنقوم بتطوير مخطط ملخص تفاعل من البداية لحالة الاستخدام "إنشاء حساب مدونة عادي جديد"، وذلك بإعادة استعمال أجزاء من مخططات التفاعل المنشأة في الفصول السابقة.

إن الاختلاف الكبير بين المثال الذي في هذا الفصل والنمذجة التي في الفصول السابقة، هو أنه يمكن مع مخطط ملخص التفاعل أن نختار من بين الأنواع المختلفة لمخطط التفاعل. باستعمال منهج مخطط ملخص التفاعل، وتتم نمذجة كل جزء من التفاعل باستعمال الأسلوب الأكثر فاعلية له.

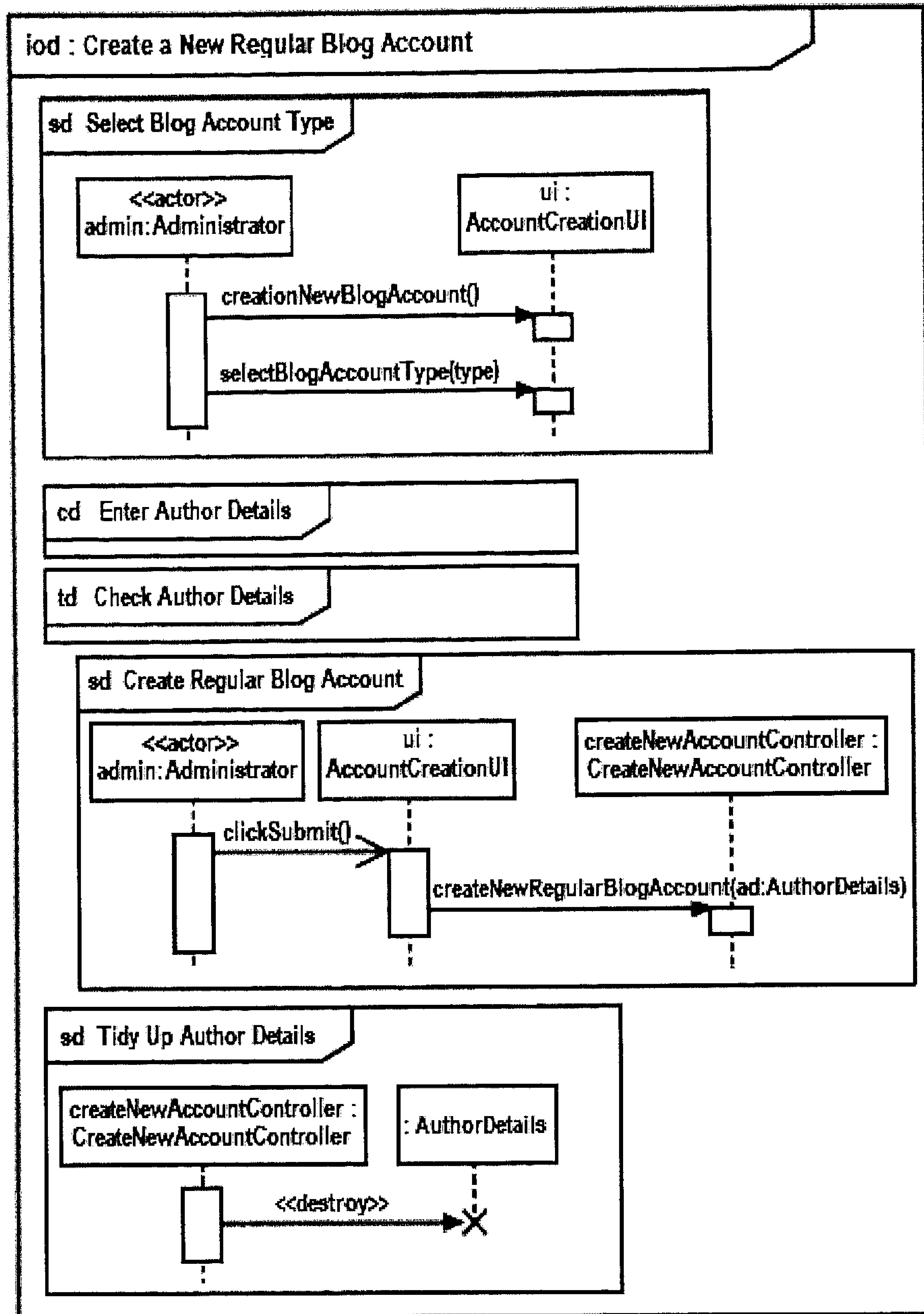
١٠-٢-١ تعاون التفاعلات Pulling Together the Interactions

نحتاج أولاً لاتخاذ القرار بخصوص كيفية تقسيم ملخّص التفاعل إلى مخططات أكثر فاعلية لكل من التفاعلات الفردية، كما هو معروض في الشكل رقم (١٠-٤).

عند نمذجة التفاعلات "اختر نوع حساب مدونة" و "إنشاء حساب مدونة عادي جديد" و "ترتيب تفاصيل الكاتب"، يكون عامل ترتيب الرسالة أكثر أهمية من أي عامل آخر. ويمكن إعادة استعمال الخطوات المتعلقة بالموضوع من مخططات التتابع المنمذجة في الفصل السابع، كما هو معروض في الشك رقم (١٠-٥).

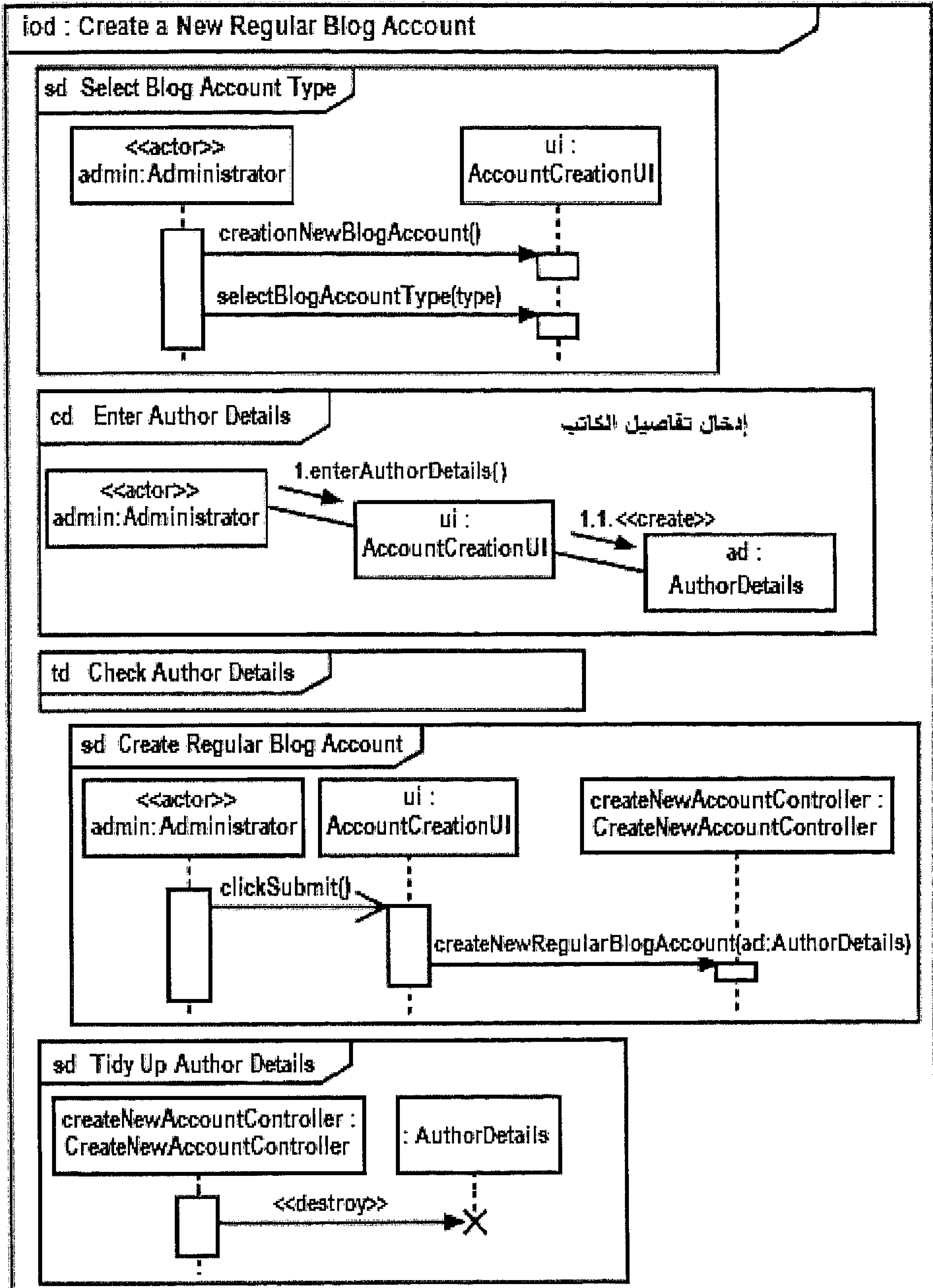


شكل رقم (١٠-٤) يتم استعمال كل الأنواع الثلاثة لمخططات التفاعل في هذه الملخّصات، وتشير iod إلى مخطط ملخّص التفاعل، وتشير sd إلى مخطط التتابع، وتشير cd إلى مخطط الاتصال، وتشير td إلى مخطط التوقيت.



شكل رقم (١٠-٥) تتم نمذجة بعض التفاعلات بشكل أفضل باستعمال مخططات التتابع للتركيز على ترتيب الرسالة.

بقصد التنويع، سيتم عرض التفاعل "إدخال تفاصيل الكاتب" كمخطط اتصال، كما هو معروض في الشكل رقم (١٠-٦).



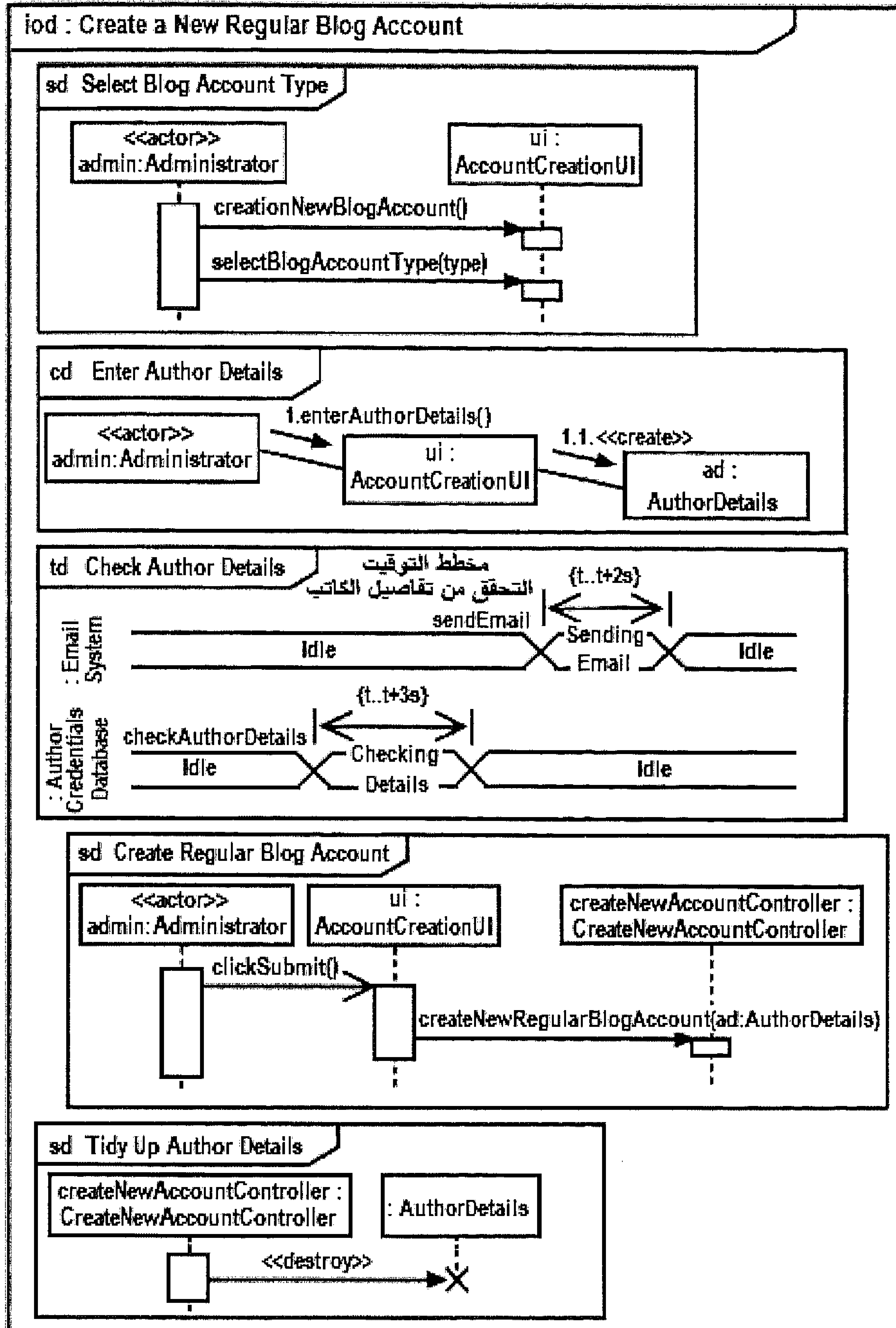
شكل رقم (١٠-٦) نمذجة التفاعل "إدخال تفاصيل الكاتب" كمخطط اتصال.

ربما تقرر تمثيل التفاعل "إدخال تفاصيل الكاتب" كمخطط اتصال بسبب سهولة فهمه، لكن هناك شبه كبير بين مخطط التتابع ومخطط الاتصال حيث إنه لا يلاحظ عادة خلطهما في ملخّص تفاعل واحد؛ ويميل المنمذجون إلى تفضيل استعمال نوع واحد منهما.

ويجب على التفاعل "التحقق من تفاصيل الكاتب" وضع القيد الزمني قيد التنفيذ حيث سيتم تنفيذ كل رسائله خلال خمسة ثوان (انظر إلى الفصل التاسع). إن التركيز في هذا الجزء من ملخّص التفاعل على التوقيت، بسبب حقيقة إمكانية احتواء ملخّص التفاعل على أي نوع من الأنواع المختلفة لمخطط التفاعل، ويمكن استعمال مخطط توقيت لأجل التفاعل "التحقق من تفاصيل الكاتب"، كما هو معروض في الشكل رقم (١٠-٧).

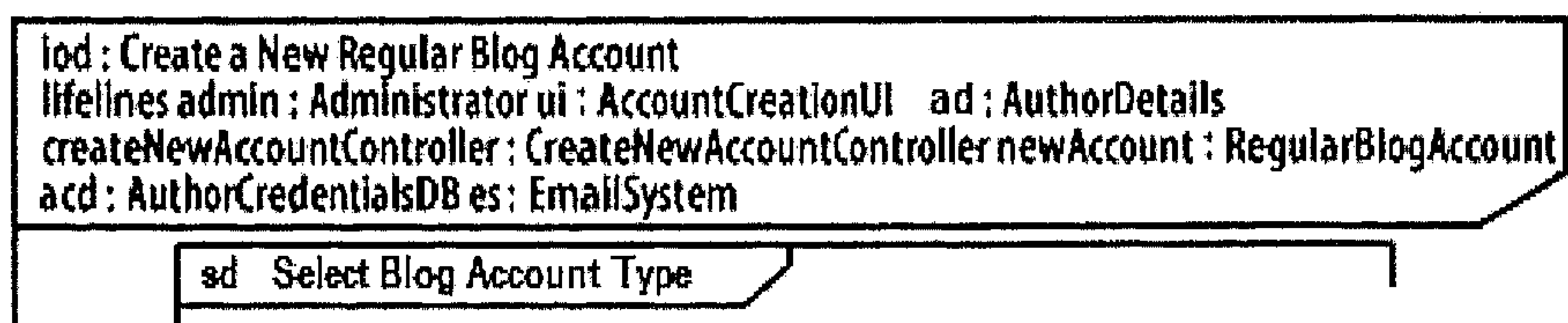


يوفر مخطط ملخّص التفاعل مكاناً مثالياً لاستعمال الترميز البديل لمخطط التوقيت، كما في الشكل رقم (١٠-٧). بما أن ملخّصات التفاعل قد تصبح كبيرة جداً، يعمل الترميز البديل بشكل أفضل إجمالاً بسبب استعماله الجيد للمساحة المتوفرة.



شكل رقم (١٠-٧) إضافة مخطط توقيت لعرض القيود الزمنية الحرجة لتفاعل واحد ضمن ملخّص التفاعل.

الآن بعد إضافة كل التفاعلات إلى ملخّص التفاعل، أصبح كل المشاركين المعنيين معروفين، ويمكن بالتالي إضافة أسمائهم إلى عنوان المخطط، كما هو معروض في الشكل رقم (١٠-٨).

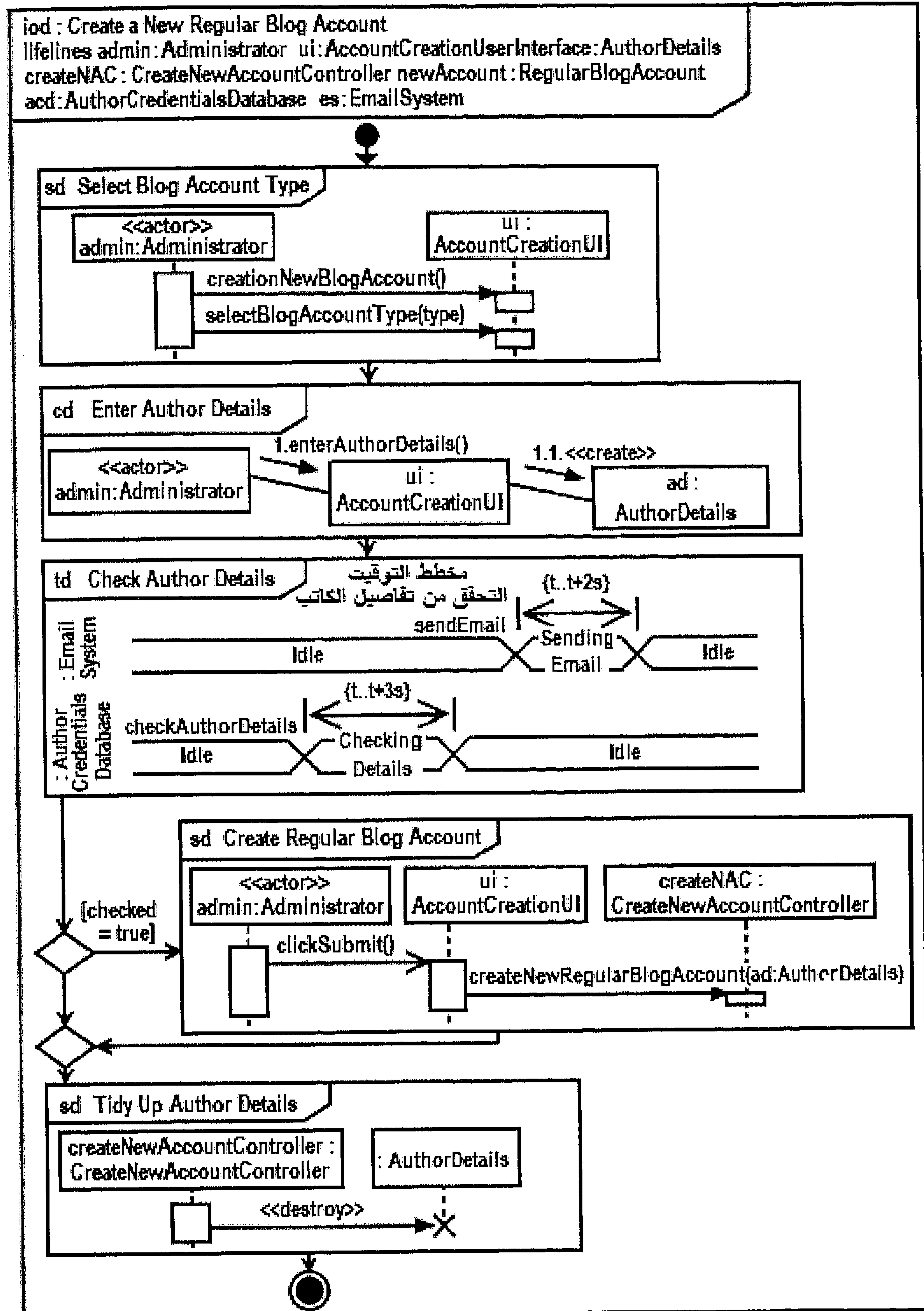


شكل رقم (١٠-٨) إضافة كل المشاركين المعنيين في التفاعل إلى قائمة خط الحياة في شريط عنوان ملخّص التفاعل.

١٠-٢-٢ ربط التفاعلات معاً Gluing the Interactions Together

إن الجزء الأخير من المعضلة في ملخّص التفاعل "إنشاء حساب مدونة عادي جديد"، هو التدفق الحالي للتحكم بين كل مخطط من مخططات التفاعل المنفصلة، كما هو معروض في الشكل رقم (١٠-٩).

يعرض تدفق التحكم في الشكل رقم (١٠-٩) بأنه يتم تنفيذ كل تفاعل من التفاعلات المنفصلة بالترتيب. إن الانحراف الوحيد عن المسار الطبيعي يحدث عند التفاعل "إنشاء حساب مدونة عادي جديد"، المعروض كمخطط تتابع، الذي يتم تنفيذه إذا تم التحقق من تفاصيل الكاتب فقط أثناء التفاعل "التحقق من تفاصيل الكاتب".



شكل رقم (١٠-٩) البدء بعقدة بداية و الانتهاء بعقدة نهاية، إن تدفق التحكم هو المسلك الذي يربط كل هذه التفاعلات معاً.

١٠-٣ ما هي الخطوة التالية؟

تـربط مخططـات ملخّـص التفاعـل مزيجـاً مـن مخططـات التتـابع والاتصـال والتوقيـت معـاً لعـرض الوصـف العـالي المسـتوى. لقـد أتممنـا هنـا اعتبـارات مخططـات التفاعـل، لكـن إذا لم يكـن واضـحاً أي مـن هـذه المخططـات يمكـن الرجـوع للـوراء ومراجعـة الفصـل الخـاص بهـا. لقـد تم تغطية مخططات التتابع في الفصل السابع، وتم وصف مخططات الاتصـال في الفصل الثامن، وتم تغطية مخططات التوقيت في الفصل التاسع.

نمذجة الهيكل الداخلي للصنف: الهياكل المركبة

MODELING A CLASS'S INTERNAL STRUCTURE: COMPOSITE STRUCTURES

تكـون المخططـات الأساسـية في لغـة النمذجـة الموحـدة (مثـل مخططـات الأصـناف و التتـابع) أحيانـاً غـير ملائمـة بشـكل كامـل لأسـر بعض التفاصيل الخاصة بالنظام. وتساعد الهياكل المركبة على ملء بعض من تلك الفجوات، وتبين كيف تنشئ الكائنات تصوراً عاماً. وتنمذج الهياكل المركبة كيف تعمل الكائنات معاً داخل صنف ما أو كيف تنجز هدفاً ما. وتعتبر الهياكل المركبة مفاهيماً متقدمة لكن من الجيد أن تكون في جعبتك لأنها تناسب بعض حالات النمذجة الخاصة بشكل تام، كما هو معروض:

الهياكل الداخلية Internal structures

تعرض الأجزاء الموجودة داخل الصنف و العلاقات التي بينها؛ يسمح لك هذا بعرض العلاقات المتأثرة بالسياق context-sensitive، أو العلاقات المندرجة ضمن سياق صنف حاوٍ.

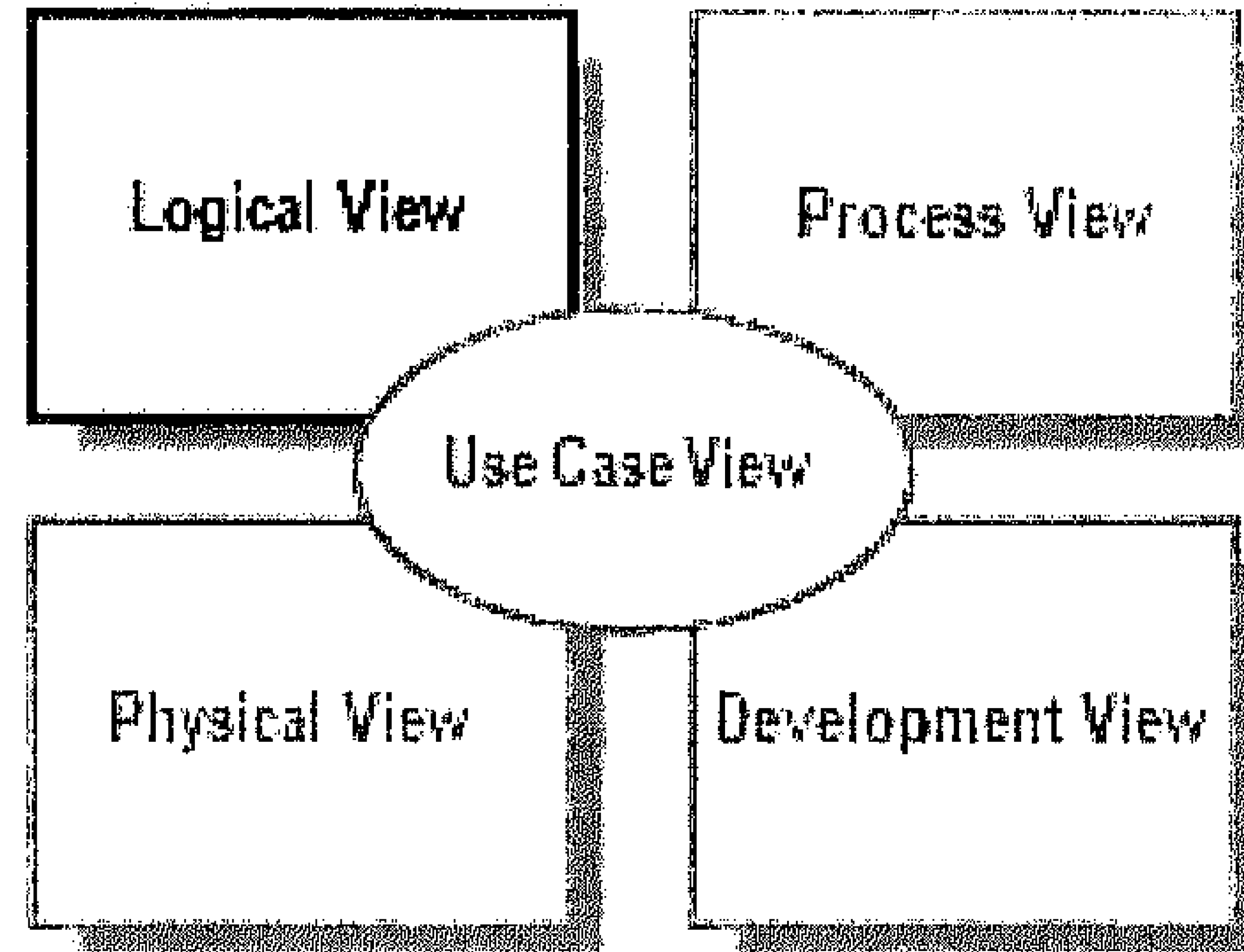
المنافذ Ports

يعرض كيفية استعمال صنف ما في النظام مع المنافذ.

التعاون Collaborations

يعرض تصميم الأنماط في النظام و على وجه العموم الكائنات المتعاونة لإنجاز هدف ما.

توفر الهياكل المركبة منظراً لأجزاء النظام وتشكل جزءاً من المنظور المنطقي لنموذج النظام، كما هو معروض في الشكل رقم (١١-١).



شكل رقم (١١-١) يأسر المنظور المنطقي التوصيفات المجردة لأجزاء النظام بما فيها الهياكل المركبة.

١١-١ الهيكل الداخلي Internal Structure

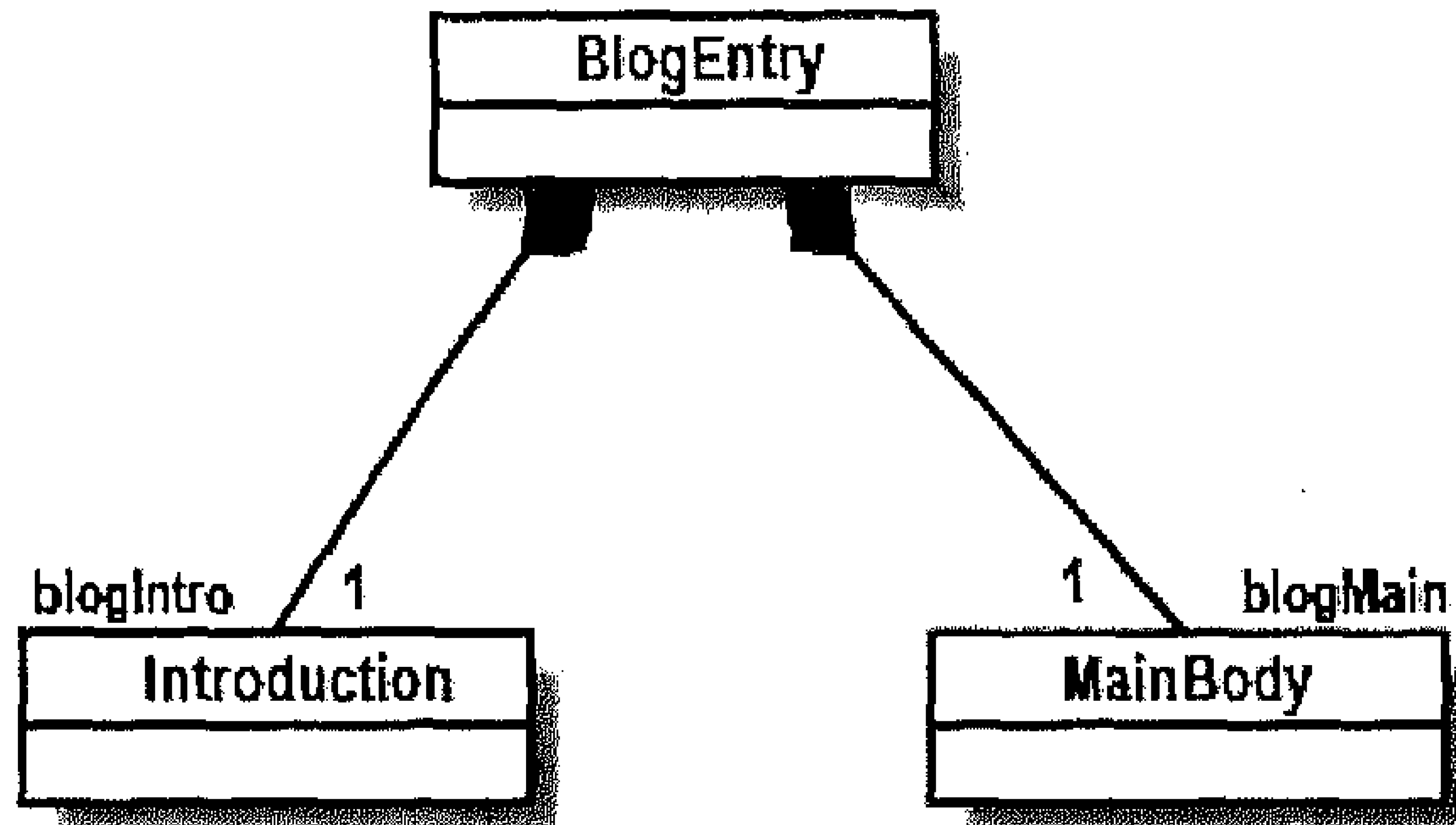
قدم الفصل الخامس العلاقات المتعلقة بمفهوم الملكية بين الأصناف، بما فيها علاقة الشراكة ("له has a") و علاقة التركيب ("يحتوي على contains a"). وتقدم الهياكل المركبة وسيلة بديلة لعرض هذه العلاقات؛ عندما تعرض الهيكل الداخلي لصنف ما، ترسم العناصر

التي يمتلكها الصنف مباشرة داخله. ويتم الاحتفاظ بالعلاقات التي بين عناصر الهيكل الداخلي للصنف داخل سياق الصنف فقط، لذلك يمكن التخيل بأنها كالعلاقات المتأثرة بالسياق. لإدراك فائدة الهياكل الداخلية، دعنا نتفحص علاقة لا يستطيع نمذجتها مخطط الأصناف.

١١-١-١ متى لا تعمل مخططات الأصناف

When Class Diagrams Won't Work

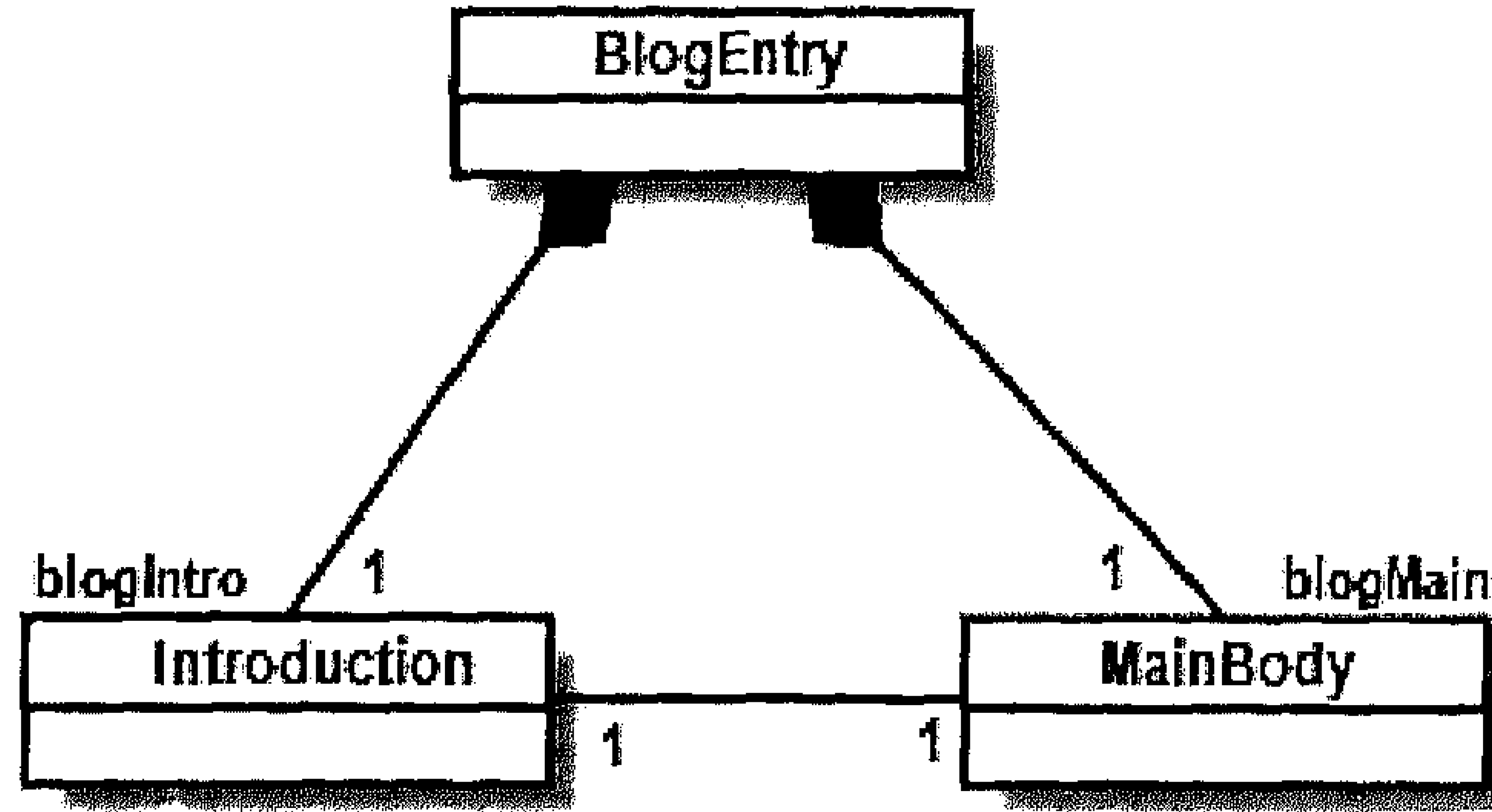
يكرر الشكل رقم (١١-٢) مخطط أصناف من الفصل الخامس، و ذلك بعرض كائن BlogEntry يحتوي على كائن مقدمة Introduction وكائن جسم رئيسي MainBody من خلال علاقة التركيب.



شكل رقم (١١-٢) يعرض مخطط الأصناف احتواء كائن **BlogEntry** على كائن **Introduction** وكائن **MainBody**.

افترض أنك تريد تحديث مخططاتك لتعبر عن أن مقدمة تدوينه لها مرجع إلى جسمها الرئيسي، لأنه من المناسب للكائنات الأخرى أن تسأل كائن مقدمة Introduction عن الكائن جسم رئيسي MainBody الذي يقدمه. كأول تغيير، ويمكن تعديل مخطط الأصناف في الشكل رقم

(١١-٢) برسم علاقة شراكة بين الصنف Introduction و الصنف MainBody، كما هو معروض في الشكل رقم (١١-٣).

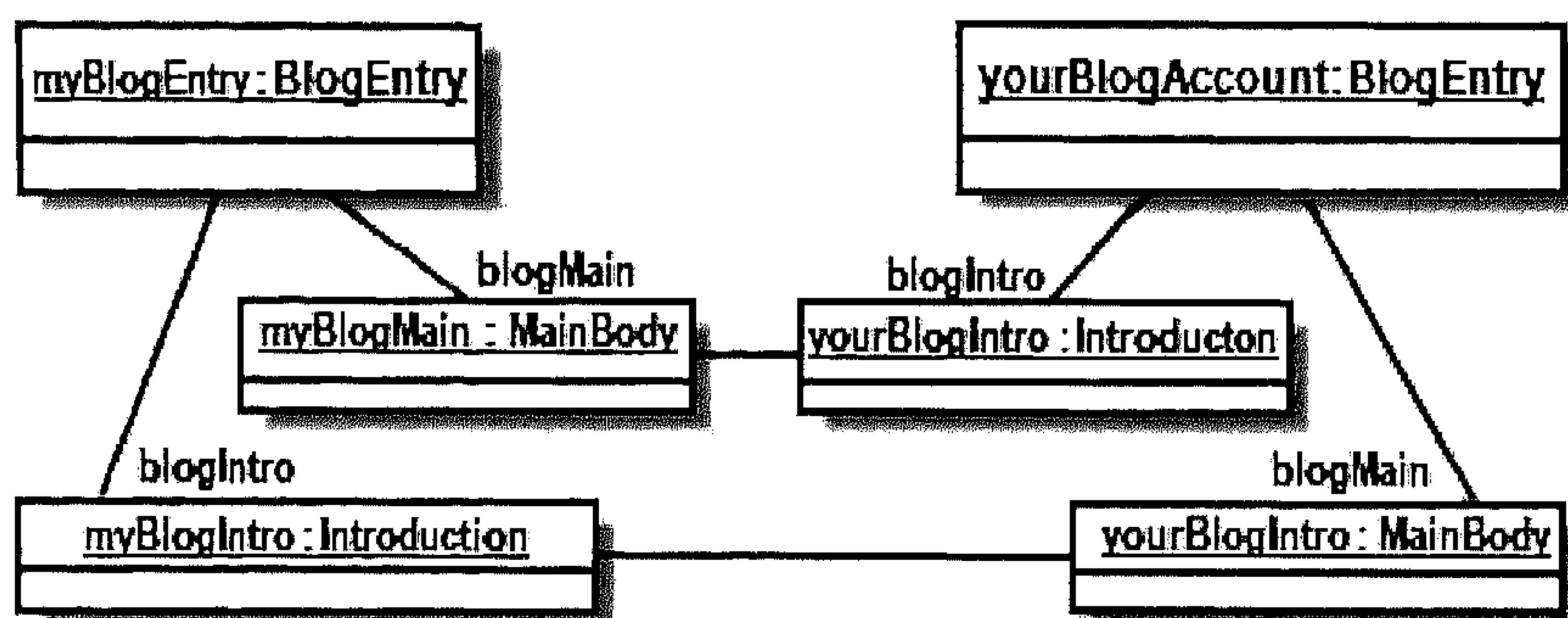


شكل رقم (١١-٣) لا يعمل التغيير الأول لعرض أن مقدمة تدوينه تقدم جسمها الرئيسي بشكل تام.

هناك مشكلة في التغيير السابق. يحدد الشكل رقم (١١-٣) أنه سيكون لدى كائن من النوع Introduction مرجع إلى كائن من النوع MainBody، لكن قد يشير هذا المرجع إلى أي كائن MainBody (ليس بالضبط الكائن MainBody المملوك من قبل نفس المثيل BlogEntry). هذا لأن الشراكة بين Introduction و MainBody معرفة لكل المثيلات من تلك الأنواع. وبشكل غير رسمي، لا يهتم كائن Introduction بنفسه بعلاقة التركيب التي بين BlogEntry و MainBody؛ مهما كان قدر اهتمام الكائن Introduction، فكل ما عليه عمله هو الارتباط بكائن MainBody (لكنه لا يهتم بأي كائن يرتبط).

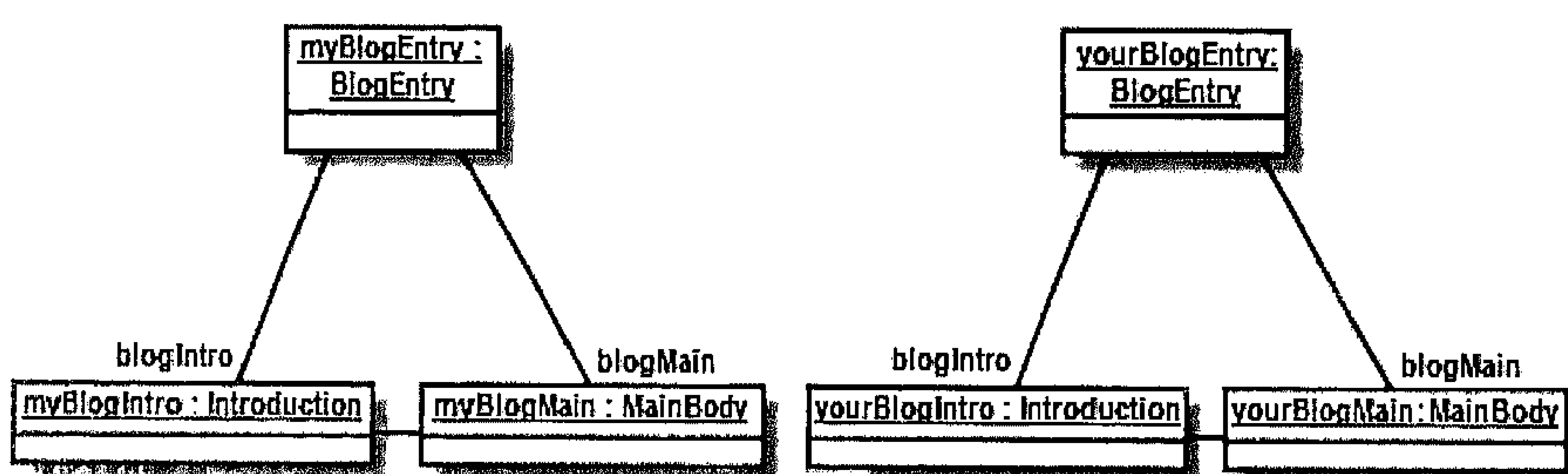
بما أن الشكل رقم (١١-٣) لا يحدد أي كائن Introduction وكائن MainBody يجب أن يرتبطا معاً، فيتوافق مخطط الكائنات الذي

في الشكل رقم (١١-٤) مع مخطط الأصناف الذي في الشكل رقم (١١-٣).



شكل رقم (١١-٤) هيكل كائنات غير مقصود لكنه صحيح.

وفقاً لمخطط الأصناف في الشكل رقم (١١-٣)، من القانوني تماماً لمقدمة تدوينة ما أن تقدم الجسم الرئيسي لتدوينة أخرى. لكن المقصود قوله أن المقدمة تقدم الجسم الرئيسي الذي يحتويه نفس الصنف الذي يحتويها، كما هو معروض في الشكل رقم (١١-٥).



شكل رقم (١١-٥) هذا هو الهيكل المقصود للكائنات.

لقد أصبح واضحاً أن مخططات الأصناف ليست جيدة للتعبير عن كيفية ربط العناصر التي يحتويها الصنف بعضها ببعض. من هنا تأتي

أهمية الهيكل الداخلي الذي يسمح بتحديد العلاقات في سياق الصنف الذي يحتويها.

قد تفكر الآن أن هذا الاختلاف يتطلب القليل من العناية. وإذا كنت تكتب شفرة إنجاز هذه الأصناف، فيمكن ضمان ربط الكائنات الصحيحة بعضها ببعض. ويمكنك أيضاً استعمال مخطط التتابع بلغة النمذجة الموحدة لعرض إنشاء الكائنات و كيفية ربطها. لكن ليبقى في ذهنك أن ترميز الهيكل الداخلي يشكل وسيلة سهلة و مناسبة لعرض العلاقات بين العناصر المُتضمَّة، خصوصاً عندما يكون لديها علاقات معقدة.



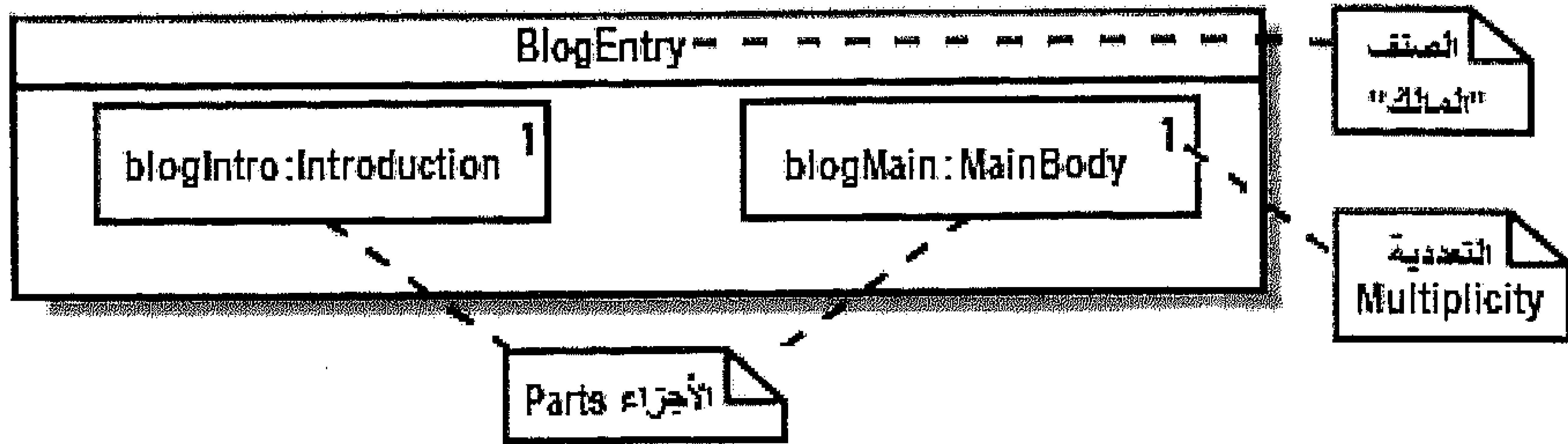
هذا مجرد مثال عن كيفية نمذجة الهياكل المركبة للعلاقات الصعبة العرض في مخططات الأصناف. اذهب إلى الموقع http://www.jot.fm/issues/issue_2004_11/column5 للحصول على مناقشة أكثر تعمقاً عن الموضوع.

١١-١-٢ أجزاء الصنف Parts of a Class

يعرض الشكل رقم (١١-٦) الهيكل الداخلي للصنف BlogEntry. لقد تم الآن رسم العناصر التي يحتويها هذا الصنف بداخله مباشرة، بدلاً من ربطها بواسطة سهم ذات رأس بشكل معين معبأ.

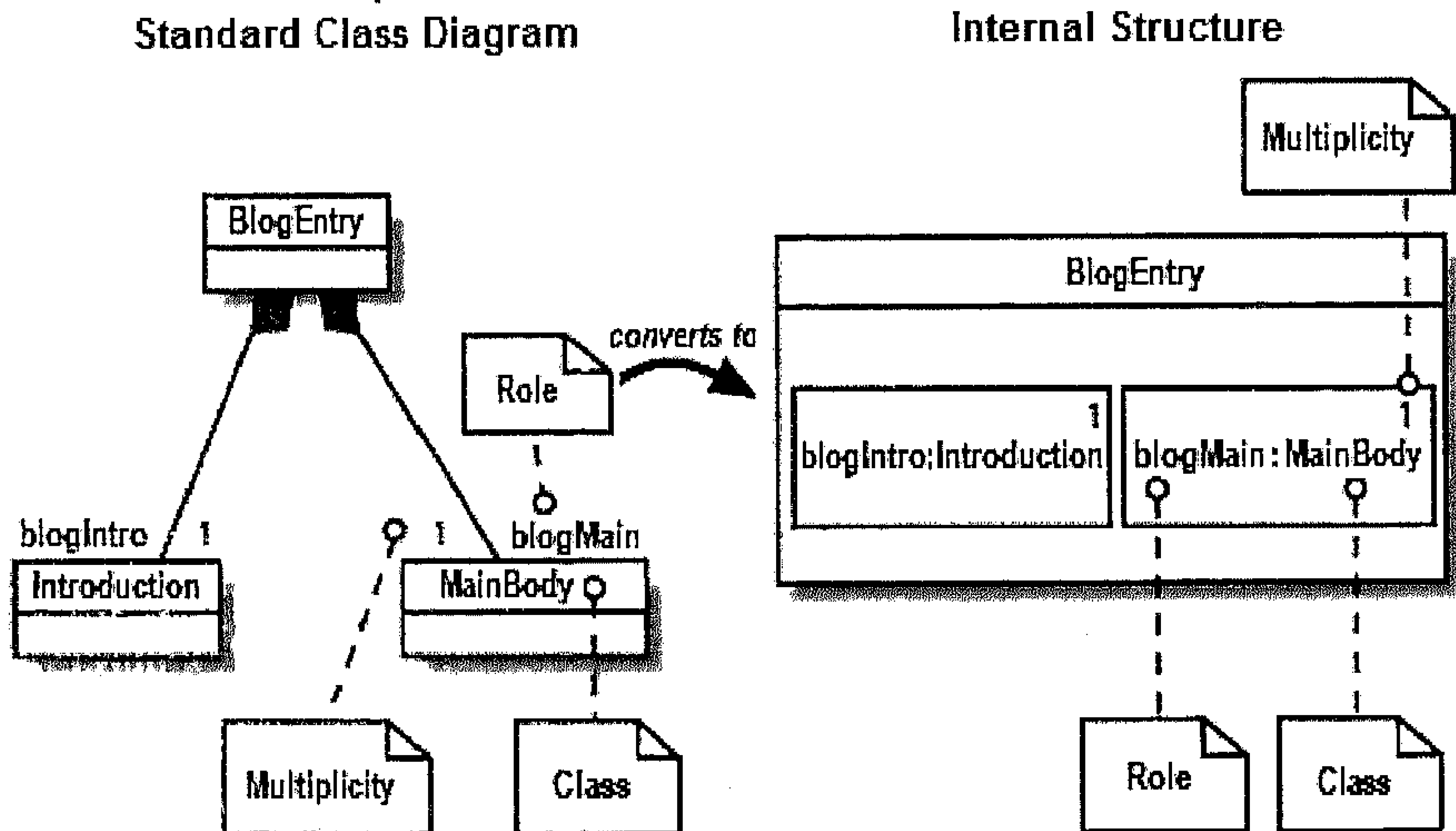
عند عرض الهيكل الداخلي للصنف تقوم برسم أجزائه، أو العناصر التي يحتويها بسبب علاقة التركيب، داخل الصنف الحاوي. يتم تحديد الأجزاء من خلال الأدوار التي تؤديها في الصنف الحاوي لها، وتكتب بالصيغة <roleName>: <type>. في الشكل رقم (١١-٦)، ولقد

تم تحديد الدور blogIntro للجزء الذي من النوع Introduction و تحديد الدور blogMain للجزء الذي من النوع MainBody. وتكتب التعددية (عدد المثيلات من ذلك الجزء) في الزاوية العليا عن اليمين داخل الجزء.



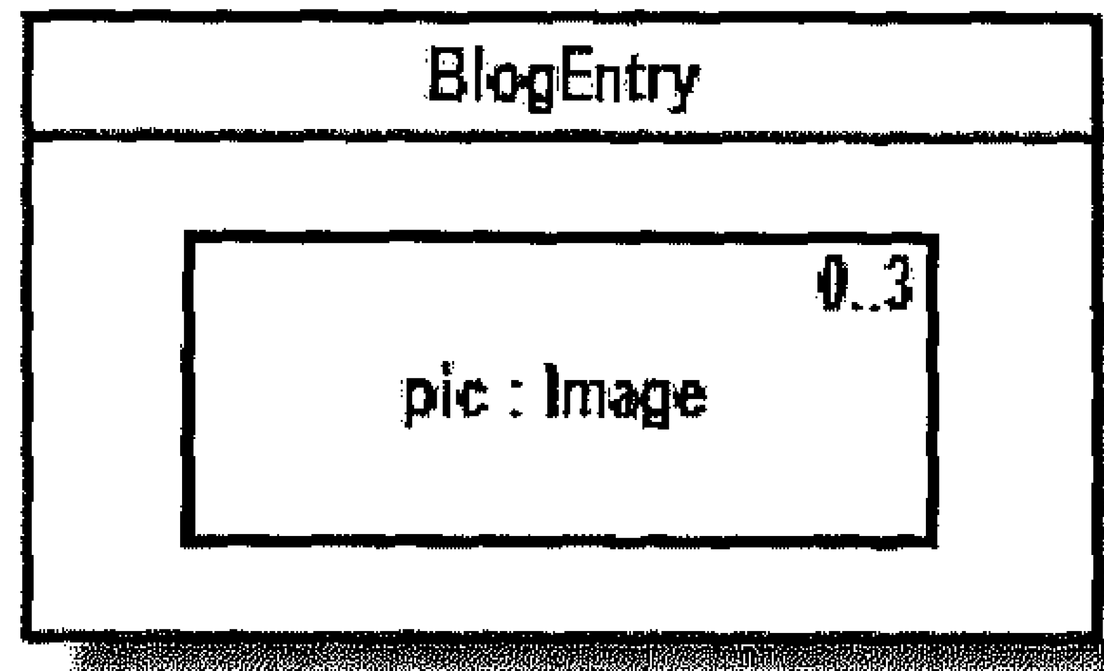
شكل رقم (١١-٦) الهيكل الداخلي للصنف BlogEntry.

ويعرض الشكل (١١-٧) مخطط الهيكل الداخلي جنباً إلى جنب مع مخطط الأصناف الذي في الشكل (١١-٢)، وذلك للسماح برؤية تطابق أسماء الأصناف والأدوار والتعدديات.



شكل رقم (١١-٧) كيفية تطابق الهيكل الداخلي للصنف BlogEntry مع مخطط الأصناف.

يبدو مفهوم الأجزاء "Parts" كالتي يحتويها الصنف BlogEntry بسيطاً، لكنه مفهوم دقيق. فالجزء عبارة عن مجموعة مثيلات قد توجد داخل كائن مثيل للصنف الحاوي لها عند وقت التشغيل. لتوضيح ذلك، قد تساعد دراسة المثال المعروض في الشكل رقم (١١-٨)، حيث للجزء ذات الدور pic تعددية من صفر إلى ثلاثة داخل الهيكل الداخلي للصنف BlogEntry.



شكل رقم (١١-٨) للجزء ذات الدور **pic** تعددية من صفر إلى ثلاثة في الهيكل الداخلي للصنف **BlogEntry**.

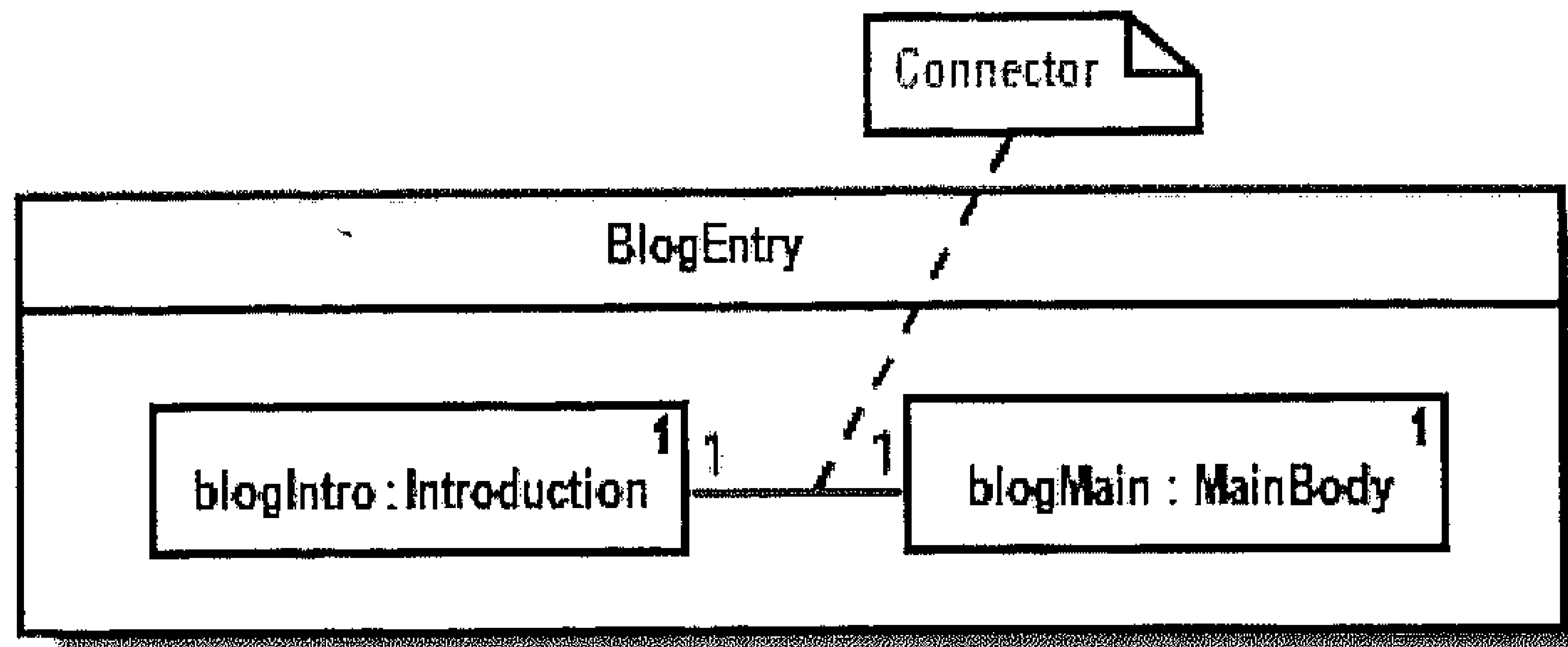
في هذه الحالة، إذا صادفت مثيلاً للصنف BlogEntry عند وقت التشغيل، سيكون لديه من صفر إلى ثلاث مثيلات من النوع Image. وقد يحتوي على كائن واحد Image، وقد يحتوي على ثلاثة كائنات Image، وهكذا، لكن ليس عليك القلق بخصوص تلك التفاصيل مع الأجزاء. ويشكل الجزء وسيلة عامة لوصف هذه الكائنات المحتواة من خلال الأدوار التي تؤديها، وذلك من دون تحديد أي كائنات حاضرة لدينا.

وبما أن الأجزاء تمثل الكائنات التي يملكها مثيل واحد من الصنف الحاوي، فيمكن تحديد العلاقات التي بين تلك الأجزاء المعنية وليس بين أي مثيلات اعتباطية من أنواع الصنف. ويعني هذا إمكانية

تحديد أن مقدمة ما تقدم الجسم الرئيسي في نفس التدوينة التي تنتمي إليها و ليس جسماً رئيسياً اعتباطياً (يتم ذلك بواسطة الروابط).

١١-١-٣ الروابط Connectors

يشار إلى العلاقات بين الأجزاء من خلال رسم رابط بين الأجزاء مع تحديد التعددية عند نهاية طرفي الرابط، كما في الشكل رقم (١١-٩).



شكل رقم (١١-٩) استعمال الروابط لربط الأجزاء في الهيكل الداخلي للصنف.

الرابط: عبارة عن موصل يسمح بالتواصل بين الأجزاء. ويعني الرابط ببساطة إمكانية تواصل مثيلات وقت التشغيل للأجزاء. ويمكن أن يكون الرابط عبارة عن مثيل وقت التشغيل لشراكة أو موصل ديناميكي تم إنشاؤه وقت التشغيل، يتم تمرير هكذا مثيل على شكل بارامتر.

ويتم تطبيق الرابط على الأجزاء المتصلة به فقط في الشكل (١١-٩)، ويعني أنه محصور بمجموعة المثيلات التي ستكون موجودة في مثيل BlogEntry. ويمكن أن تكون متأكداً الآن أن المقدمة تقدّم الجسم الرئيسي الذي في نفس التدوينة الخاصة بها.



إن الترميز المستعمل للتعددية على الروابط هو نفسه المستعمل للتعددية على الشراكات، و الذي تم مناقشته في الفصل الرابع.

١١-١-٤ ترميزات بديلة للتعددية

Alternative Multiplicity Notations

يعرض الشكل رقم (١١-٤) التعددية باستعمال عدداً في الزاوية العليا عن يمين مستطيل الجزء. كما يمكن عرض التعددية أيضاً بعد اسم و نوع الجزء بين القوسَين [ ]، كما هو معروض في الشكل رقم (١١-١٠).

blog : Blog 1   blog : Blog[1]

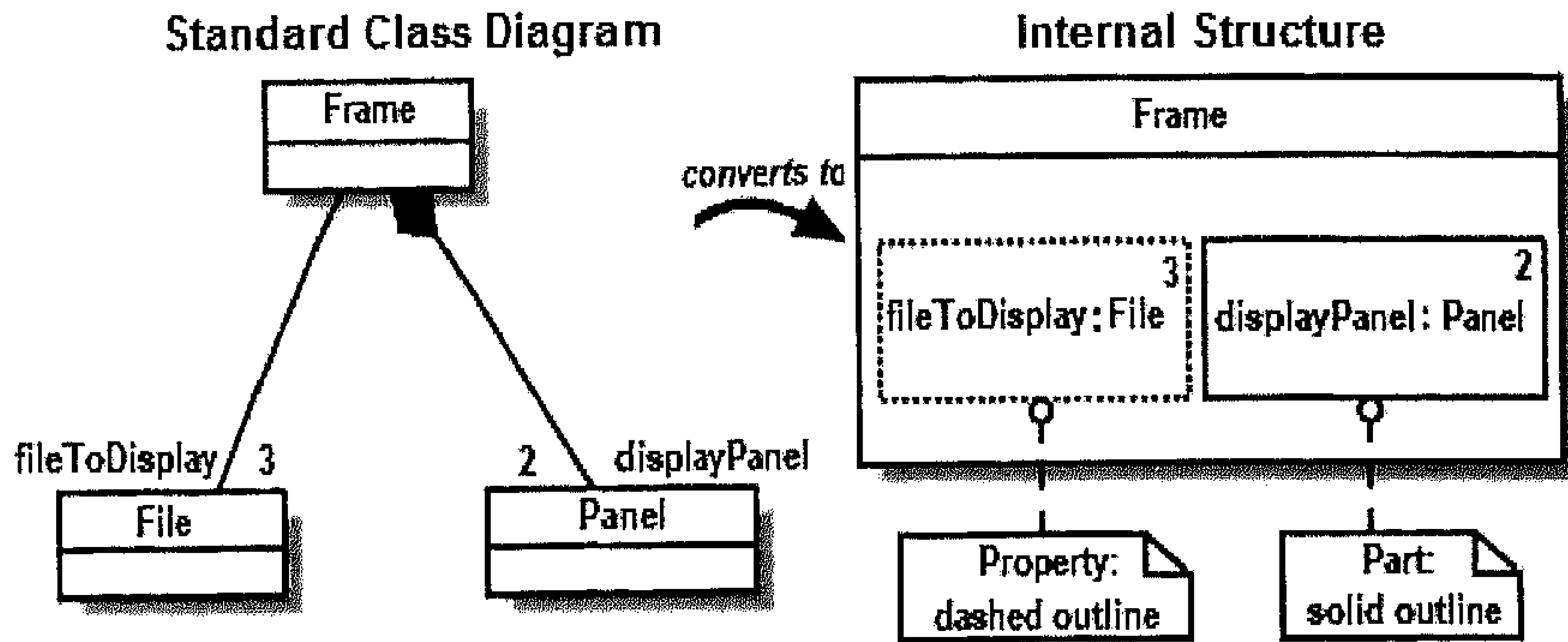
شكل رقم (١١-١٠) ترميزات متكافئة للتعددية.

١١-١-٥ الميزات Properties

بالإضافة إلى عرض الأجزاء المُتضمَّنة بواسطة علاقة التركيب، يمكن أيضاً عرض الميزات التي يشار إليها من خلال علاقة الشراكة والتي قد تكون مشتركة مع أصناف أخرى في النظام.

ويتم رسم الميزات باستعمال مستطيل منقط الأضلع، بخلاف الأجزاء التي يتم رسمها باستعمال مستطيل غير منقط. ويعرض الشكل رقم (١١-١١) مخطط أصناف يضم شراكة بين الصنف إطار Frame والصنف ملف File، وبالتالي يعرض كيف تظهر File حقاً كميزة داخل الهيكل الداخلي للصنف Frame. وينمذج الشكل رقم (١١-١١) أداة للدمج لها واجهة مستخدم رسومية GUI، والتي تعرض لوحَين panels حيث

يحتوي اللوح الأول على ملفَين للمقارنة ويحتوي اللوح الثاني على ملف الدمج.



شكل رقم (١١-١١) الأجزاء و الميزات التي في الهيكل الداخلي للصنف.

يتم استعمال نفس الترميز للأجزاء والميزات باستثناء شكل أضلع المستطيل حيث تكون منقطة مع الميزات وغير منقطة مع الأجزاء، وتقوم بتحديد الأدوار والأنواع والتعددية بنفس الأسلوب. ويمكن ربط الميزات بأجزاء أو ميزات أخرى باستعمال الروابط كما هو الحال مع الأجزاء.

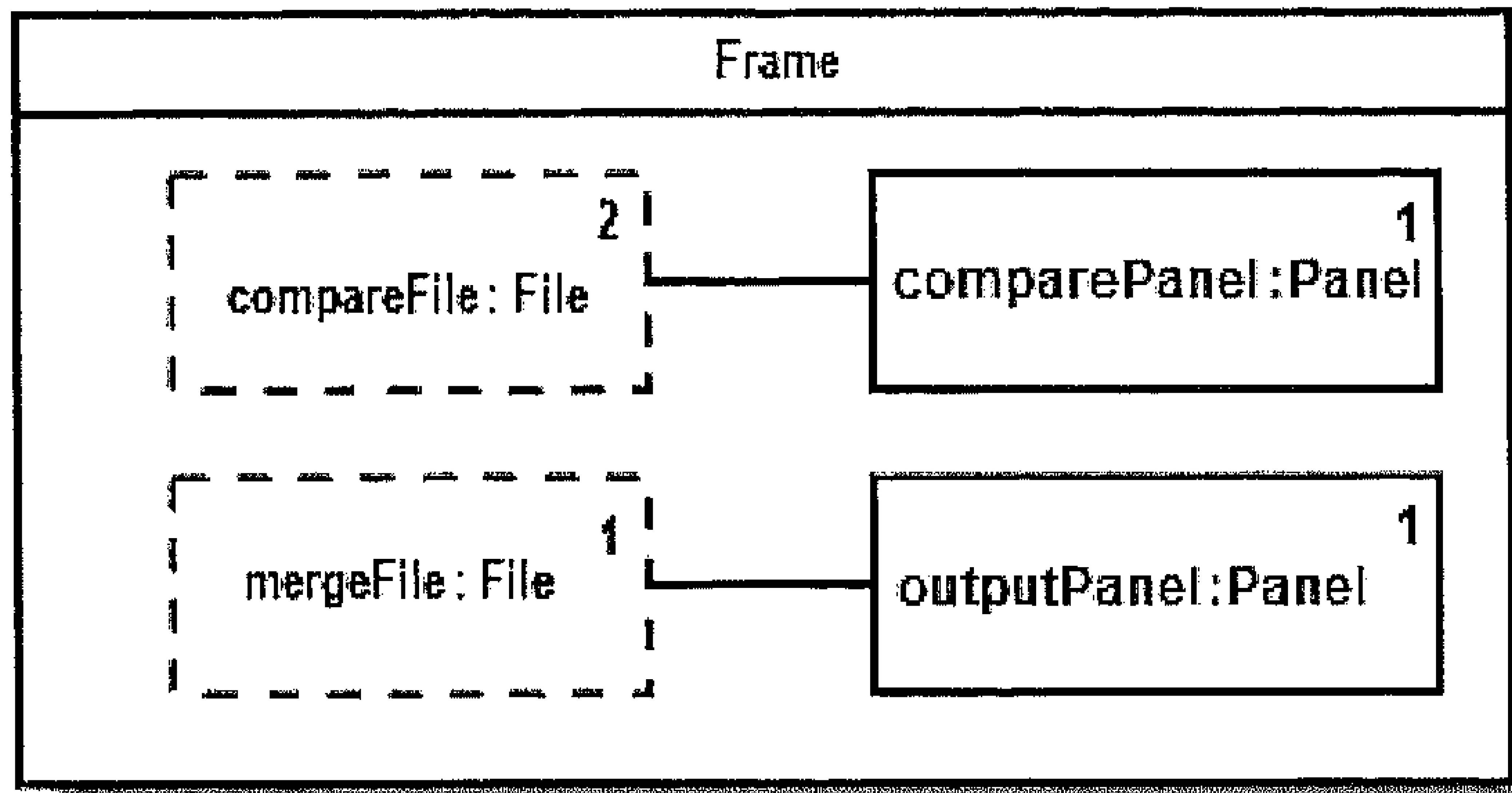
١١-١-٦ عرض علاقات معقدة بين العناصر المُحتواة

Showing Complex Relationships between Contained Items

يفيد عرض الهيكل الداخلي للصنف بخاصة عندما تتعلق العناصر التي يحتويها الصنف بعضها ببعض بأسلوب غير اعتيادي. ارجع مرة أخرى إلى مثال أداة الدمج في الشكل رقم (١١-١١)، وافترض أنك تريد أن تنمذج بشكل صريح وواضح أن اللوح الأول يعرض ملفي المقارنة ويعرض اللوح الآخر ملف الدمج. ويمكن القيام بذلك بتعريف أدوار مفصّلة أكثر

للملفات والألـواح، وذلـك لعـرض كيفيـة تعلّـق تلـك العناصـر بعضـها ببعض داخل الإطار، كما هو معروض في الشكل رقم (١١-١٢).

ويظهر الشكل رقم (١١-١٢) بوضوح لإمكانية تواجد أجزاء (أو ميزات) من نفس النوع تؤدي أدواراً مختلفة. وتساعد الهياكل الداخلية في جعل تلك الأدوار وعلاقاتها واضحة وصريحة.



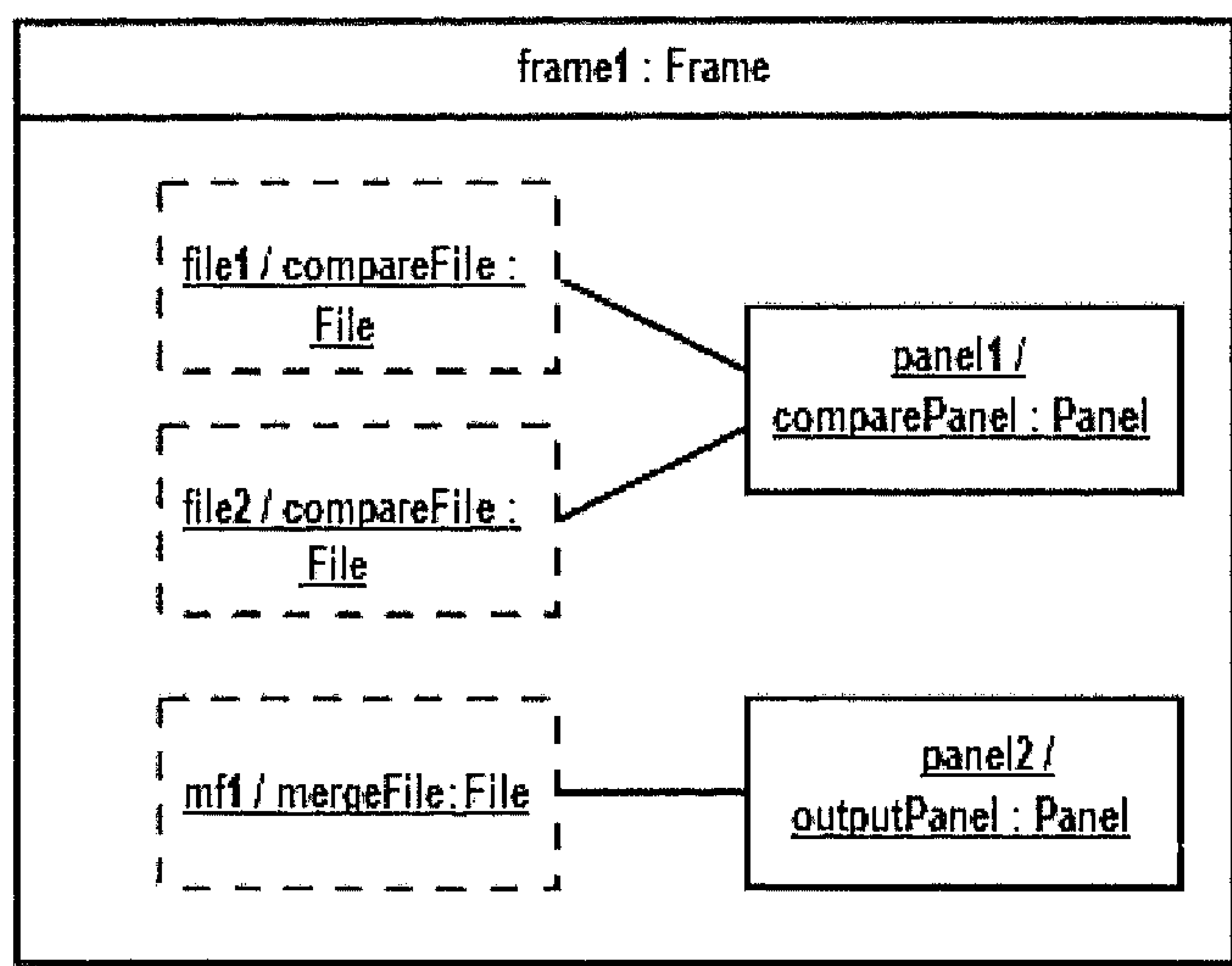
شكل رقم (١١-١٢) مخطط لهيكل داخلي أكثر تفصيلاً حيث يحدد كيفية تعلّق الملفات files والألواح panels بعضها ببعض داخل الإطار frame.

١١-١-٧ مثيلات الهيكل الداخلي Internal Structure Instances

كما يمكن نمذجة مثيلات الأصناف (تم تقديمها في الفصل السادس)، يمكن أيضاً إظهار أن مثيلات الأصناف تمتلك هيكلاً داخلياً. ويشكل هذا بشكل أساسي مخطط كائنات للأصناف ذات هياكل داخلية. كما في الفصل السادس، يسمح هذا بعرض أمثلة مهمة عن الكائنات الموجودة في النظام وقت التشغيل.

وإذا كنت تعرض مثيلاً لصنف ذي هيكل داخلي، فتكون كذلك تعرض أجزاءه وميزاته كمثيلات. قم بتحديد مثيلات الأجزاء

والميزات بكتابة الاسم متبعاً برمز الشَرطة (/)، ثم بالدور والنوع كالعادة، أي بالصيغة <type> :< <role> / {<name>}. على أية حال، بما أن تلك العناصر عبارة عن مثيلات فسيتم الآن وضع سطر تحتها. ويعرض الشكل رقم (١١-١٣) مثالاً عن مثيل وقت التشغيل لمخطط الهيكل الداخلي المعروض في الشكل رقم (١١-١٢).



شكل رقم (١١-١٣) مثيل للصنف Frame مع مثيلات عن الأجزاء التي يحتويها.

كما هو الحال مع مخططات الكائن، يسمح عرض مثيلات الأصناف ذات الهياكل الداخلي بعرض نماذج الترتيبات configurations في النظام وقت تشغيل.

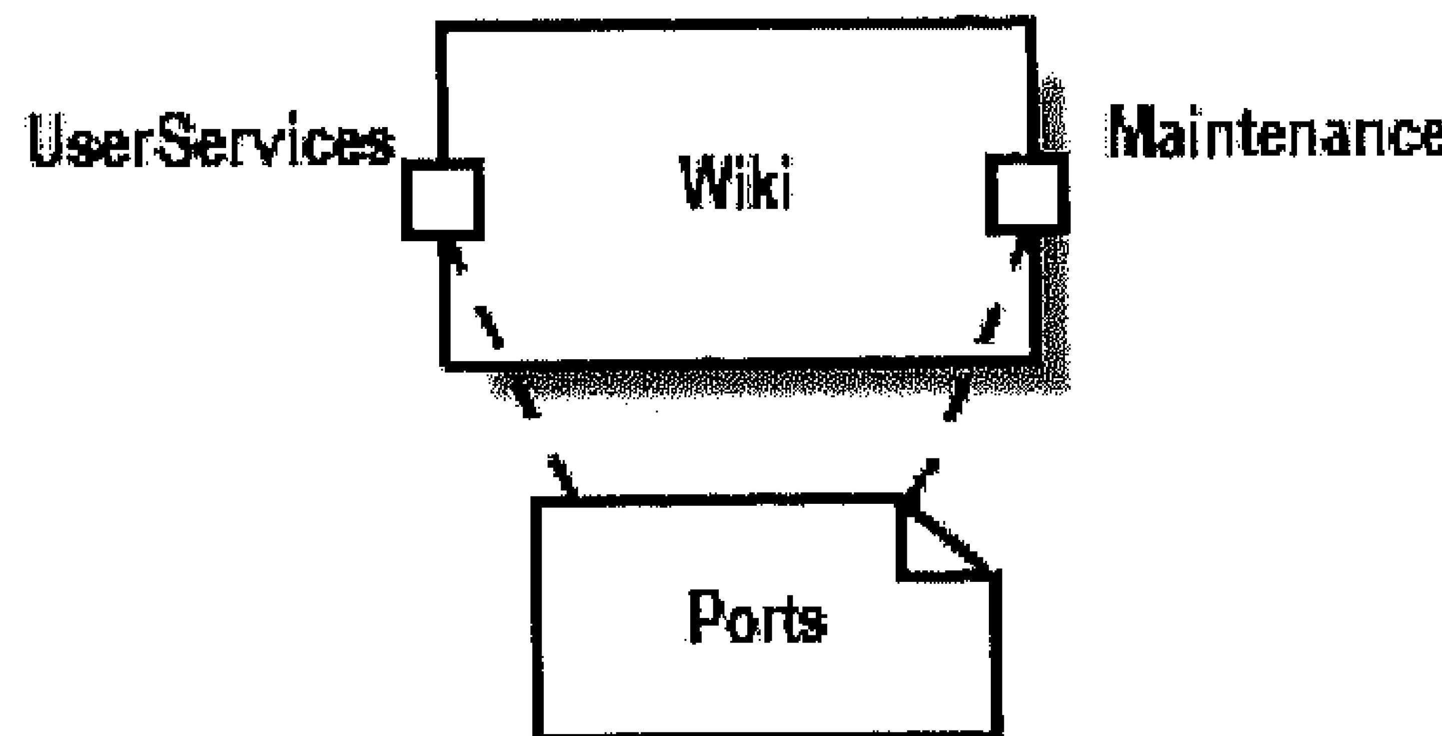
١١-٢ عرض كيفية استعمال الصنف
Showing How a Class Is Used

يركز الهيكل الداخلي للصنف على العناصر التي بداخله؛ وتركز المنافذ ports على خارج الصنف من خلال عرض كيفية استعمال الصنف من قبل أصناف أخرى.

ويشكل المنفذ نقطة تفاعل بين الصنف والعالم الخارجي. وعادة ما يمثل وسيلة مختلفة لاستعمال الصنف من قبل أنواع مختلفة من العملاء clients. على سبيل المثال، ويمكن أن يكون للصنف ويكي Wiki استعمالان مختلفان:

- السماح للمستخدمين بمعاينة الويكي وتحريره.
- توفير خدمات الصيانة للمدراء الراغبين بإنجاز أمور، مثل التراجع في الويكي عند التزويد بمحتوى خاطئ.

يتم تمثيل كل استعمال مختلف للصنف بواسطة منفذ، يتم رسم المنفذ كمستطيل صغير على إحدى حدود الصنف، كما هو معروض في الشكل رقم (١١-١٤). اكتب الاسم بجانب المنفذ لتبيان هدفه.



شكل رقم (١١-١٤) يبين الصنف Wiki ذو المنفذين أنه يوفر مهارات في خدمات المستخدم UserServices وفي الصيانة Maintenance.

من الشائع امتلاك الأصناف واجهات متعلقة بالمنافذ. ويمكن استعمال المنافذ لتجميع الواجهات ذات العلاقة لعرض الخدمات المتوفرة في ذلك المنفذ.

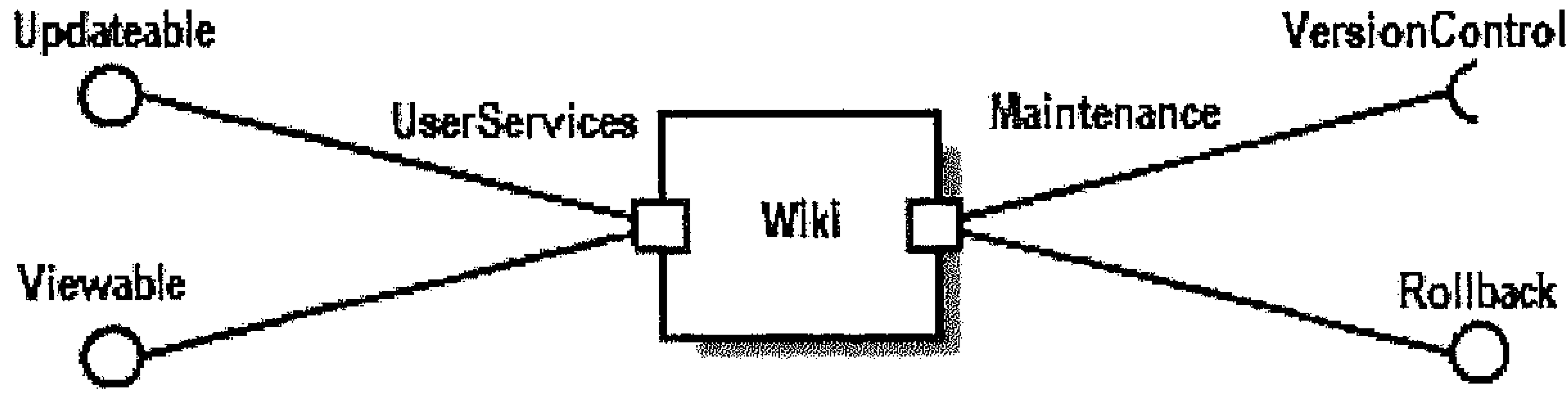
تذكر من الفصل الخامس أنه يمكن للصنف إنجاز واجهة ما، ويمكن عرض هذه العلاقة باستعمال رمز الكرة الخاص بالواجهة. عندما يقوم الصنف بإنجاز واجهة ما، تسمى هذه الواجهة بالواجهة المُتوفّرة **provided interface** من الصنف. ويمكن استعمال الواجهة المُتوفّرة من قبل الأصناف الأخرى للوصول إلى الصنف من خلال الواجهة.

وبشكل مماثل، يمكن أن يكون للأصناف واجهات مُتطلّبة **required interface**. إن الواجهة المتطلبة هي واجهة يتطلب الصنف بأن تعمل. وبشكل أدقّ، يحتاج الصنف إلى مكّون أو صنف آخر يقوم بإنجاز تلك الواجهة حتى يستطيع الصنف القيام بعمله. ويتم عرض الواجهة المتطلبة باستعمال شكل مصاصة مفتوحة أو رمز التجويف.



ويتم استعمال الواجهات المُتوفّرة والمتطلبة لتطوير الارتباط غير المحكم بين الأصناف والمكونات. وهي مهمة جداً للمكونات لذلك ستتم مناقشتها بشكل مفصل في مخططات المكونات (انظر إلى الفصل الثاني عشر).

افترض أن الصنف Wiki السابق الذي ينجز الواجهتَين قابل للتحديث Updateable و قابل للمعاينة Viewable، وذلك للسماح للأصناف الأخرى بتحديث ومعاينة الويكي من خلالهما. وترتبط هاتان الواجهتان بمنفذ خدمات المستخدم UserServices، لذلك يمكن رسمهما ممتدّتَين خارج المنفذ UserServices، كما هو معروض في الشكل رقم (١١-١٥).



شكل رقم (١١-١٥) يمكن استعمال المنافذ لتجميع "هكذا" واجهات.

يعرض الشكل رقم (١١-١٥) أن للمنفذ صيانة Maintenance واجهة مُتوفّرة تدعى التراجع Rollback تسمح للمدراء بالتراجع في الويكي Wiki. بالإضافة إلى ذلك لهذا المنفذ واجهة مُتطلبة تدعى تحكم بالإصدار VersionControl، التي هي عبارة عن خدمة يستعملها الويكي Wiki للتحكم بالإصدار.

١١-٣ عرض الأنماط مع التعاون
Showing Patterns with Collaborations

يقوم التعاون بعرض الكائنات و هي تعمل معاً - ربما بشكل مؤقت - لإنجاز أمر ما. وقد يشبه هذا مخططات الكائنات (انظر إلى الفصل السادس)، لكن يركز التعاون على أمر مختلف: وصف الكائنات من خلال الدور الذي تؤديه في سيناريو ما، وذلك بتوفير وصف نصي عالي المستوى لما تقوم بعمله الكائنات.

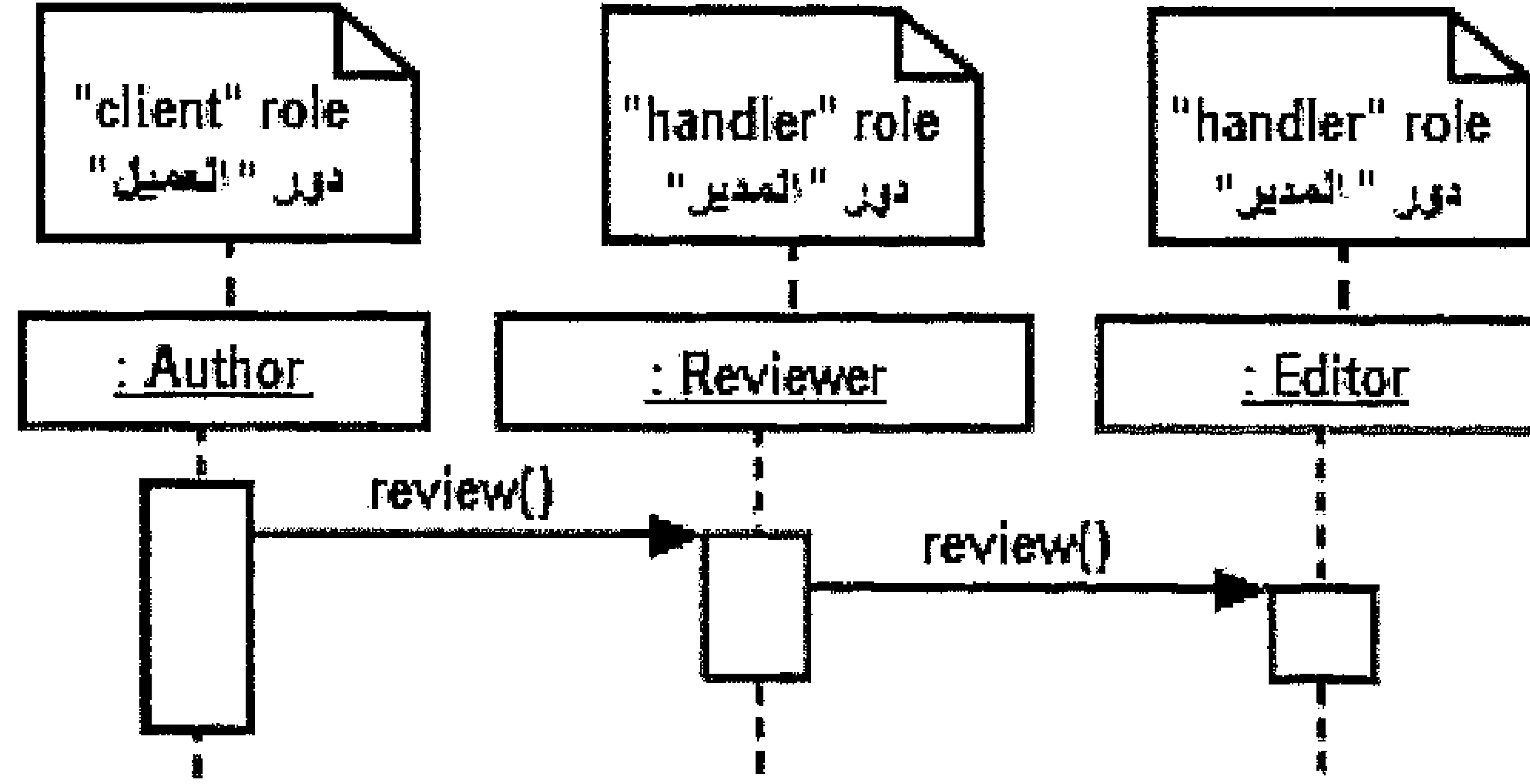
يشكل التعاون وسيلة جيدة لتوثيق أنماط التصميم **design patterns**، التي تشكل حلولاً لمشاكل شائعة في تصميم البرمجيات. حتى إذا لم تكن قد سمعت بها مطلقا، ربما تكون استعملت بعض الأنماط من دون معرفة ذلك. إن الصنف مراقب Observer والصنف قابل للمراقبة Observable في واجهات البرامج التطبيقية في جافا (Java API) عبارة عن

إنجاز لنمط التصميم Observer (وسيلة للكائن لاستقبال إشعار بتغيير كائن آخر).



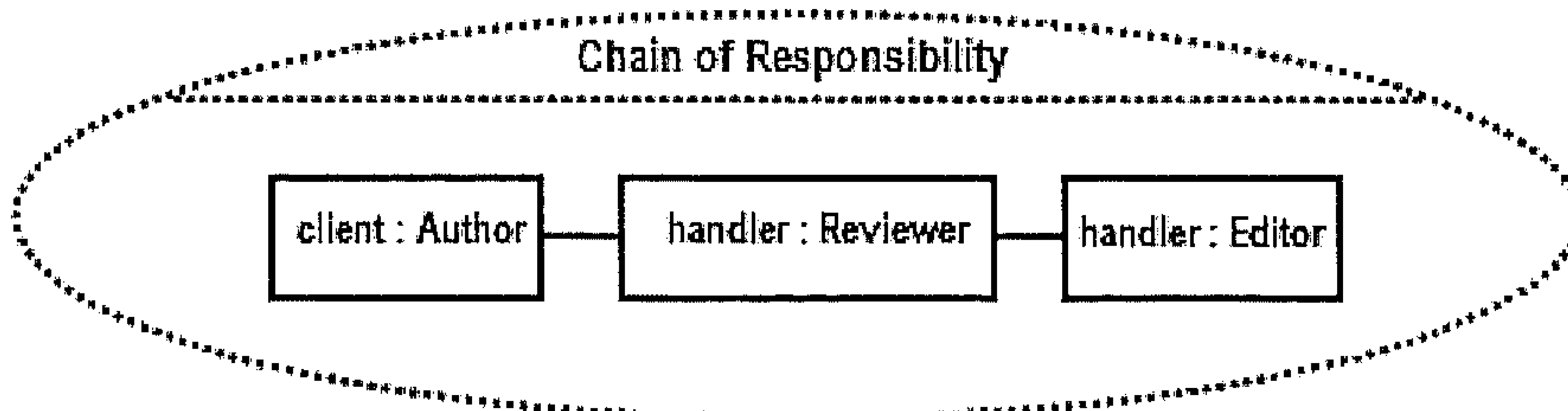
للمزيد من المعلومات عن تصميم الأنماط وإمكانية تحسينها وتصميم البرمجيات، راجع الكتاب "Design Patterns: Elements of Reusable Object Oriented Software (Addition Wesley)"، أو الكتاب "Head First Design Patterns (O'Reilly)".

دعنا ندرس مسألة ما في تصميم نظام إدارة المحتوى CMS يمكن حلها باستعمال نمط التصميم، ومن ثم سنقوم بنمذجة هذا النمط باستعمال التعاون. افترض أن النظام CMS يتطلب عملية المصادقة على المحتوى: يقدم الكاتب المحتوى، قد يرفض المراجع هذا المحتوى أو ينقله إلى المحرّر، وقد يرفض المحرّر هذا المحتوى أو يقبله. لقد قررنا إنجاز هذا التدفق باستعمال نمط التصميم سلسلة المسؤولية **Chain Of Responsibility (COR)**. يسمح نمط التصميم COR لكائن ما بإرسال طلب معيّن من دون القلق حول الكائن الذي سيعالج هذا الطلب في النهاية. في نمط التصميم COR، يقدم الزبون client الطلب و يقوم كل مسئول handler في السلسلة باتخاذ القرار بمعالجة هذا الطلب أو بتمريره إلى المسئول التالي. في عملية المصادقة على المحتوى، سيؤدي الكاتب دور الزبون، بينما يقوم كل من المراجع والمحرر بأداء دور المسئول. ويقوم مخطط التتابع في الشكل رقم (١١-١٦) بتوضيح هذا التدفق. ارجع إلى الفصل السابع لإنعاش ذاكرتك بخصوص مخططات التتابع.



شكل رقم (١١-١٦) يعرض مخطط التتابع كيفية استعمال نمط التصميم سلسلة المسؤولية COR في عملية المصادقة على المحتوى.

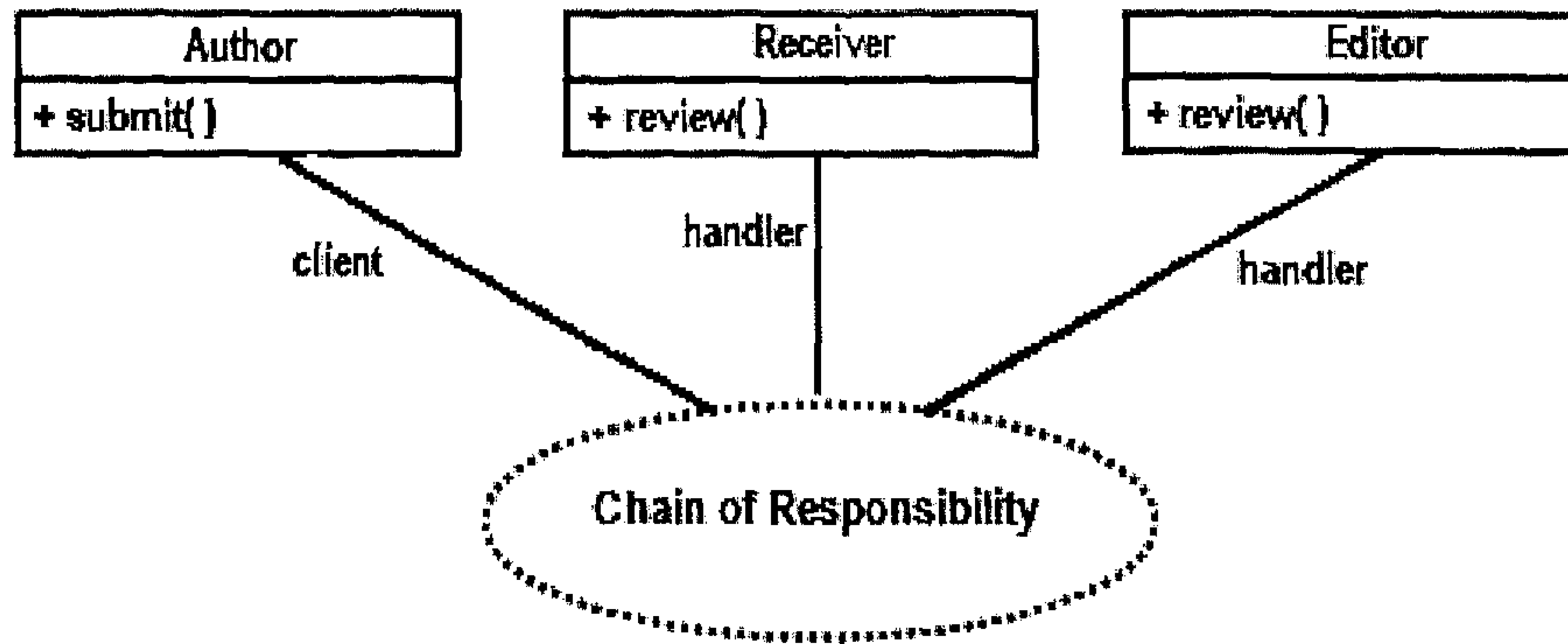
هناك أسلوبان لنمذجة هذا النمط باستخدام التعاون. يستعمل الأسلوب الأول شكلاً بيضاوياً كبيراً منقطاً مع رسم المشاركين بالتعاون بداخله. نقوم بتسمية المشارك بالارتكاز على الدور الذي يؤديه في التعاون، وعلى نوع صنفه أو واجهته، ويكتب بالصيغة <role>: <type>. ويتم ربط المشاركين معاً باستخدام الروابط لعرض كيفية تواصلهم معاً. يكتب اسم التعاون بأعلى الشكل البيضاوي فوق خط منقط. يعرض الشكل رقم (١١-١٧) التعاون سلسلة المسؤولية COR باستعمال الترميز الأول. في هذا التعاون COR، ويؤدي المشارك من النوع كاتب Author دور الزبون Client، ويؤدي المشاركون الآخرون دور المسئول handler.



شكل رقم (١١-١٧) يعرض التعاون نمط التصميم سلسلة المسؤولية COR في عملية المصادقة على المحتوى.

يمكنك التفكر بالمشاركين في التعاون كأنهم مكان احتواء للكائنات، لأنه وقت التشغيل ستملأ الكائنات تلك الأماكن (أو تلعب الأدوار). تكون الروابط وصلات مؤقتة؛ تعني الروابط أن كائنات وقت التشغيل ستتواصل أثناء التعاون، لكن ليس عليها التواصل خارج التعاون.

إن الأسلوب الآخر لرسم التعاون معروض في الشكل رقم (١١-١٨). في هذا الترميز، ويتم عرض مستطيلات أصناف (أو واجهات) المشاركين بربط كل واحد منهم بالشكل البيضاوي الصغير للتعاون. اكتب أدوار المشاركين على طول خطوط الربط. ويفيد هذا الترميز بعرض تفاصيل الصنف أو الواجهة كعملياتها.

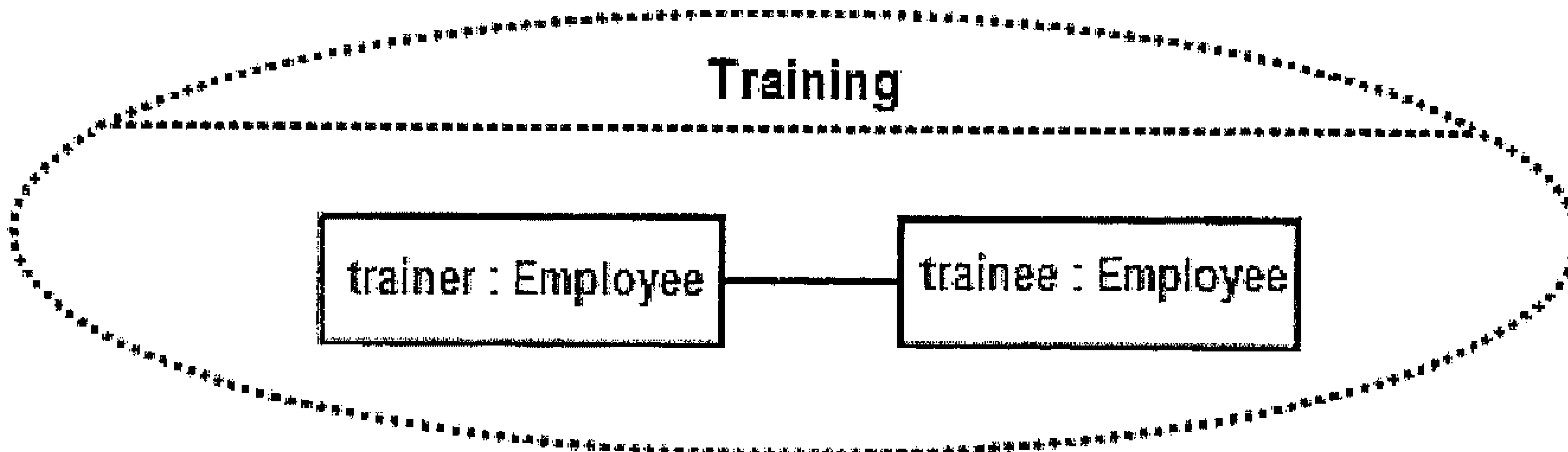


شكل رقم (١١-١٨) تمثيل بديل لنمط تصميم سلسلة المسؤولية COR.

قد لا يبدو التعاون مفيداً جداً، لكن تكمن قوته في قدرته على التعبير عن الأنماط التي قد لا تكون واضحة في المخططات الأخرى، كمخططات الأصناف أو التتابع. عند غياب التعاون، يجب علينا التفكير

بوصف ما يجري بأسلوبنا الخاص، كاستعمال الملاحظات الملحقة التي في الشكل رقم (١١-١٦).

وبما أن التعاون عبارة عن علاقات مؤقتة فقط، فهو يتمتع ببعض ميزات وقت التشغيل المهمة التي يتم وصفها بشكل أفضل من خلال استعمال مثال شائع عن التعاون. افترض أنه لدى شركة ما جلسات تدريبية شهرية حيث يتغير الموضوع في كل جلسة، ويقوم في كل جلسة الخبير المقيم المختص بالموضوع بإنجاز التدريب. تتم نمذجة هذا الأمر كالتعاون تدريب Training الذي يكون لديه مشاركين يلعبون دور المدرب trainer ودور المتدرب trainee، كما هو معروض في الشكل رقم (١١-١٩).



شكل رقم (١١-١٩) يبين التعاون تدريب Training أن الكائنات المشتركة في تعاون ما عند وقت التشغيل يمكن أن تتفاعل مع تعاونات مختلفة بأساليب مختلفة.

دعنا نتوجه الآن إلى بعض الكائنات التي قد تؤدي هذه الأدوار وقت التشغيل. لنعتبر "بَنْ Ben" خبيراً في لغة XML، لذلك أثناء التعاون "تدريب XML"، يؤدي " بَنْ" دور المدرب ويؤدي زميله في العمل "بول Paul" دور المتدرب. على أية حال، لنعتبر "بول" خبيراً في لغة جافا، لذلك أثناء

تعاون التدريب جافا، يؤدي "بول" دور المدرب ويؤدي "بَنْ" دور المتدرب. يوضح مثال التدريب النقاط التالية:

١- لا يكون الكائن محصوراً بدوره في التعاون حيث بإمكانه أداء أدوارٍ مختلفة في تعاونات مختلفة. يبقى الموظف " بَنْ" و الموظف "بول" نفس الكائنَين إنما يؤديان فقط أدواراً مختلفة في تعاونات تدريب مختلفة.

٢- إن الكائنات التي في التعاون غير مملوكة من قبله و قد تكون موجودة قبله أو بعده. تعيش الكائنان " بَنْ" و "بول" خارج التدريب.

٣- رغم ارتباط الكائنات التي في التعاون بعضها ببعض، فليس من الضروري تواصلها خارج التعاون فيما بينها. قد لا يتحدث " بَنْ" و"بول" الواحد مع الآخر ما لم يكن عليهما ذلك قطعاً أثناء جلسات التدريب.



يبيّن التعاون "تدريب" أيضاً أنه يمكن استعمال التعاون لوصف أي نوع من تفاعل الكائن، والذي يمكن تلخيصه بشكل رائع باستعمال جملة قصيرة (ليس باستعمال أنماط التصميم فقط).

يعتبر هذا ترميزاً خاصاً بلغة النمذجة الموحدة، لأن الشكل البيضاوي يلفت الانتباه ببساطة إلى وجود نمط ما، ويتم وصفه بشكل موجز و بعبارات عالية المستوى. لكن يعتبر التعاون ذو قيمة لهذا السبب بالضبط. وتدور أنماط التصميم حول إيجاد مفردات مشتركة بين مطوري البرمجيات لحل المسائل الشائعة، ويساعد التعاون في تبادل تلك المفردات. لا يبين التعاون التفاعلات التفصيلية كالرسائل الممررة بين الكائنات

كما تفعله مخططات التتابع والاتصال، ولكن يمكن أن يفيد ذلك عندما يتعلق الأمر بالتعبير بشكل موجز عن نمط معروف جيداً.

١١-٤ ما هي الخطوة التالية؟

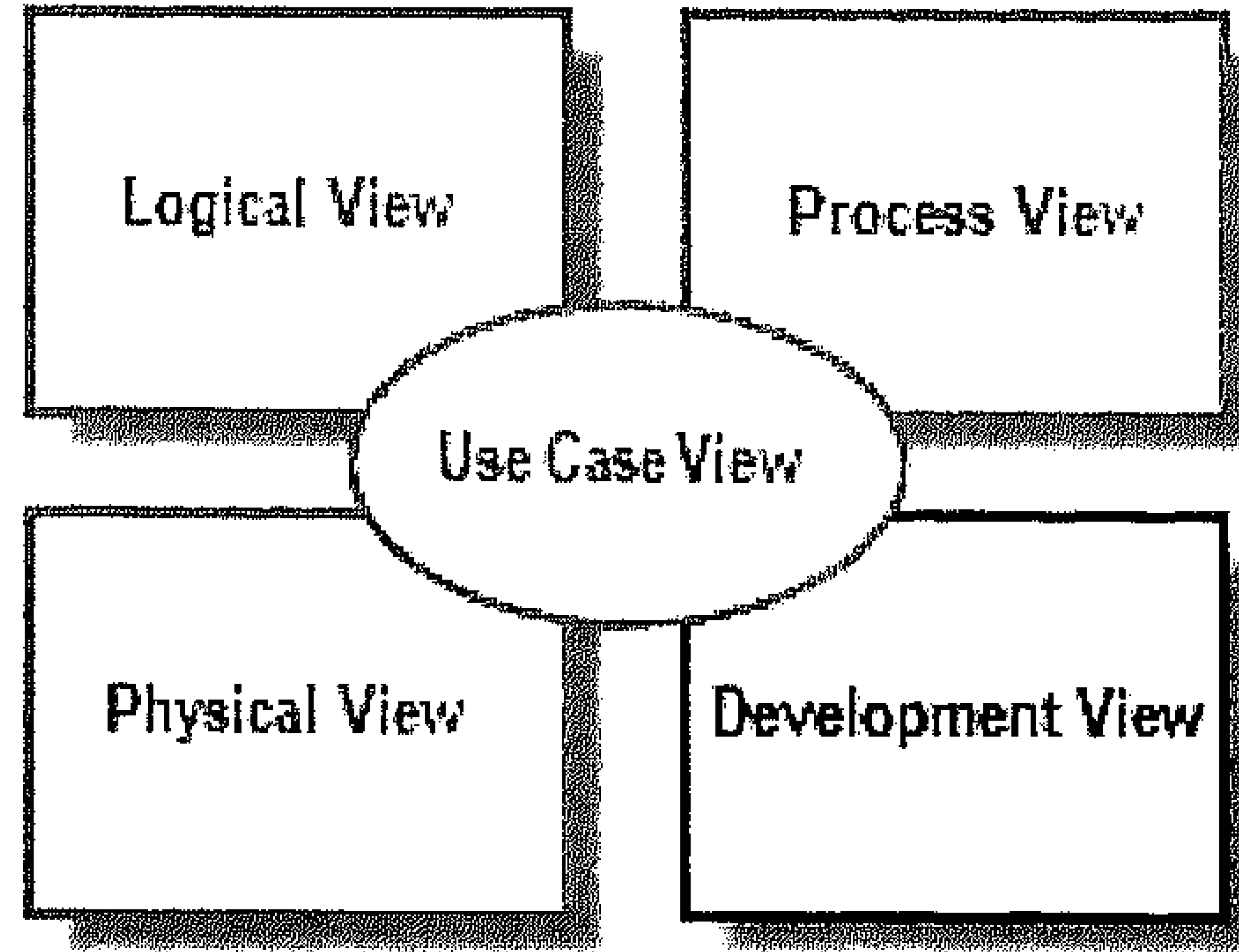
إن مفهومي المنافذ و الهيكل الداخلي للصنف (اللذين تم تقديمهما في الهياكل المركبة)، سيعاد استعمالهما بصعوبة مع المكوّنات components في مخططات المكوّنات. وتسمح مخططات المكونات بعرض المكونات الرئيسية (الأجزاء القابلة لإعادة الاستعمال) في النظام. عادة ما تشكل المكونات العناصر الرئيسية في هندسة المعمارية، وذلك باستعمال أصناف أخرى لإنجاز سلوكياتها، لهذا السبب يكون الهيكل الداخلي مهماً جداً للمكونات. وعادة ما تستعمل المنافذ لعرض الأساليب البدائية لاستعمال المكوّنات. وستتم تغطية مخططات المكوّنات في الفصل الثاني عشر.

إدارة أجزاء النظام وإعادة استعمالها: مخططات المكوّنات

MANAGING AND REUSING YOUR SYSTEM'S PARTS: COMPONENT DIAGRAMS

عند تصميم نظام برمجي، من النادر الانتقال مباشرة من متطلبات النظام إلى تعريف الأصناف التي فيه. ومع الأنظمة غير البديهية، من المفيد التخطيط للأجزاء عالية المستوى في النظام، وذلك من أجل بناء معمارية النظام وإدارة التعقيد complexity والاعتمادية dependencies بين أجزائه. ويتم استعمال المكوّنات لتنظيم النظام على شكل أجزاء برمجية قابلة لإدارة و إعادة الاستعمال والتبادل.

تقوم مخططات المكوّنات في UML بنمذجة المكوّنات التي في النظام و بالتالي فهي تشكل جزءاً من منظور التطوير، كما هو معروض في الشكل رقم (١٢-١). ويصف منظور التطوير كيفية تنظيم أجزاء النظام على شكل وحدات و مكوّنات، وهي مهمة للمساعدة على إدارة الطبقات داخل معمارية النظام.



شكل رقم (١٢-١) يصف منظور التطوير للنموذج كيفية تنظيم أجزاء النظام على شكل وحدات و مكوّنات.

١٢-١ ما هو المكوّن؟
What Is a Component?

يشكل المكوّن جزءاً مُغلّفاً من البرنامج قابلاً للاستبدال وإعادة الاستعمال. ويمكن التفكير بالمكوّنات كأنها قطع البناء: نقوم بجمع المكوّنات لتتلاءم معاً حتى تشكّل البرنامج (وربما لبناء مكونات أكبر بشكل تراكمي). لهذا السبب، يتراوح نطاق حجم المكوّنات من الصغير نسبياً (بحجم الصنف تقريباً) وحتى النظام الفرعي الكبير.

وتشكل عناصر التشغيل الرئيسي التي ستستعمل كثيراً في النظام مرشحاً جيداً لتكون مكوّنات. تعتبر البرامج مثل المسجّل logger ومحلل لغة الترميز الموسعة XML parser، أو عربات التسوق عبر الإنترنت مكوّنات قد تكون قمت باستعمالها سابقاً. ويصادف أن تكون تلك الأمور عبارة عن أمثلة شائعة لمكوّنات الطرف الثالث، غير أن نفس المبادئ تنطبق على المكوّنات التي تنشئها بنفسك.

في نظامك الخاص، قد تقوم بإنشاء مكوّن يوفر الخدمات أو إمكانية الوصول إلى البيانات. على سبيل المثال، قد يكون لديك مكوّن

"إدارة التحويل" ضمن نظام إدارة المحتوى يقوم بتحويل المدونات إلى صيغ مختلفة، مثل ملقّم أو تغذية آر اس اس RSS feeds (RSS هو اختصار لمصطلح Really simple syndication وهي خدمة لنقل الأخبار والمواضيع الأخرى). وعادة ما يتم استعمال ملقّم RSS لتزويد المحتويات المتوفر على الإنترنت (مثل المدوّنات).

يمكن لمكوّن في UML أن يعمل نفس الأمور التي بإمكانه عملها الصنف كعلاقات التعميم و المشاركة مع الأصناف والمكوّنات الأخرى وإنجاز الواجهات وامتلاك العمليات، وهكذا. من ناحية أخرى، كما هو الحال مع الهياكل المركبة (انظر إلى الفصل الحادي عشر)، ويمكن أن يكون لدى المكوّنات منافذ وأن تقدم هيكلاً داخلياً. إنّ الاختلاف الأساسي بين الصنف والمكوّن - عموماً - هو امتلاك المكوّن مسؤوليات أكبر من التي يمتلكها الصنف. على سبيل المثال، قد تقوم بإنشاء صنف خاص بمعلومات المستخدم يحتوي على معلومات الاتصال به (الاسم وعنوان البريد الإلكتروني)، وتقوم بإنشاء مكوّن لإدارة المستخدم يسمح بإنشاء حسابات المستخدمين والتحقق من أصالتها. من ناحية أخرى، من الشائع احتواء المكوّن لأصناف أو مكوّنات أخرى واستعمالها للقيام بعمله.

بما أن المكوّنات هي عناصر رئيسية في تصميم البرنامج، فمن المهم أن تكون مقترنة بشكل ضعيف loosely coupling كي لا يؤثر التغيير في مكوّن ما على باقي النظام. ومن أجل تعزيز الاقتران الضعيف و مبدأ التغليف، يتم الوصول إلى المكوّنات من خلال الواجهات. تذكر من الفصل الخامس قيام الواجهات بفصل السلوك عن كيفية إنجازه. وعند السماح بوصول المكوّنات بعضها إلى بعض من خلال الواجهات، يمكن

تقليل فرصة أن يتسبب تغيير ما في مكوّن واحد بموجة توقفات في مجمل النظام. (ارجع إلى الفصل الخامس لمراجعة الواجهات).

١٢-٢ مكوّن أساسي في لغة النمذجة الموحدة
A Basic Component in UML

يتم تمثيل المكوّن باستعمال مستطيل مع الحاشية <<component>> و أيقونة اختيارية عند الزاوية العليا عن اليمين، حيث ترسم هذه الأيقونة كمستطيل بعروة. يعرض الشكل رقم (١٢-٢) المكوّن إدارة التحويل **ConversionManagement** المستعمل في نظام إدارة المحتوى، والذي يقوم بتحويل المدونات إلى صيغ مختلفة وتوفير معلومات التلقيم، مثل RSS feeds.

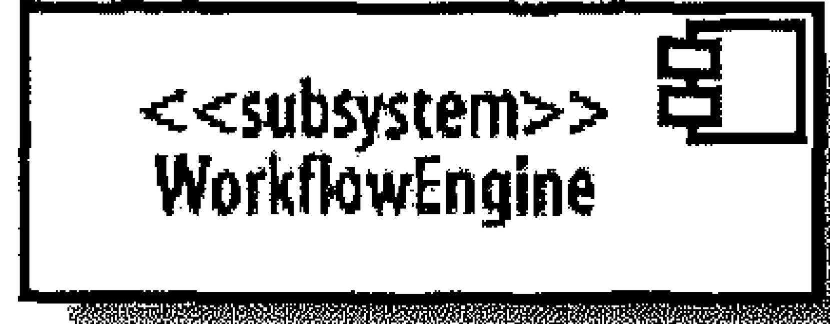


شكل رقم (١٢-٢) يقوم الرمز الأساسي للمكوّن بعرض المكوّن **ConversionManagement**.

في النسخ السابقة للغة النمذجة الموحدة، كان رمز المكوّن عبارة عن نسخة مكبرة عن شكل أيقونة المستطيل بعروة، لذلك لا تتفاجأ إذا ما زالت بعض أدوات لغة النمذجة الموحدة لا تزال تستعمل هذا الترميز.

يمكننا عرض أن المكوّن عبارة عن نظام فرعي لنظام كبير جداً من خلال استبدال الحاشية <<component>> بالحاشية <<subsystem>>، كما هو معروض في الشكل رقم (١٢-٣). ويشكل النظام الفرعي نظاماً ثانوياً أو تابعاً حيث يكون جزءاً من نظام أكبر. وتعتبر لغة النمذجة

الموحدة النظام الفرعي كنوع خاص من المكوّنات وهي مرنة بخصوص كيفية استعمالنا لهذه الحاشية، لكن من الأفضل الاحتفاظ بها لاستعمالها مع الأجزاء الكبرى في النظام الشامل، مثل نظام قديم يقوم بالتزويد بالبيانات أو محرّك تدفق عمل في نظام إدارة المحتوى.



شكل رقم (١٢-٣) يمكنك الاستبدال بالحاشية <<subsystem>> لعرض الأجزاء الكبرى في النظام.

١٢-٣ الواجهات المُتوفّرة و الواجهات المُتطلّبة للمكوّن
Provided and Required Interfaces of a Component

تحتاج المكوّنات إلى أن تكون مقترنة بشكل ضعيف، ليصبح بإمكانها التغيّر من دون الإلزام بإجراء تغييرات على أجزاء أخرى من النظام، من هنا تأتي أهمية الواجهات. وتتفاعل المكوّنات فيما بينها من خلال الواجهات المُتوفّرة والواجهات المُتطلّبة للسيطرة على التبعيّات بين المكوّنات و لجعل المكوّنات قابلة للتبادل.

إن الواجهة المُتوفّرة الخاصة بالمكوّن هي الواجهة التي يقوم المكوّن بإنجازها. تتفاعل المكوّنات والأصناف الأخرى مع المكوّن من خلال الواجهات المُتوفّرة الخاصة به. وتصرّح الواجهة المُتوفّرة عن الخدمات التي سيوفرها المكوّن.

إن الواجهة المُتطلّبة الخاصة بالمكوّن هي واجهة يحتاج إليها المكوّن لكي يشتغل. وبشكل أدقّ، يحتاج المكوّن إلى صنف أو مكوّن آخر يقوم بإنجاز تلك الواجهة لكي يشتغل بدوره. ولكن للبقاء مع هدف

الاقتران الضعيف، يقوم المكوّن بالوصول إلى الصنف أو المكوّن الآخر من خلال الواجهة المُتطلّبة. وتصرح الواجهة المُتطلّبة عن الخدمات التي سيحتاج إليها المكوّن.

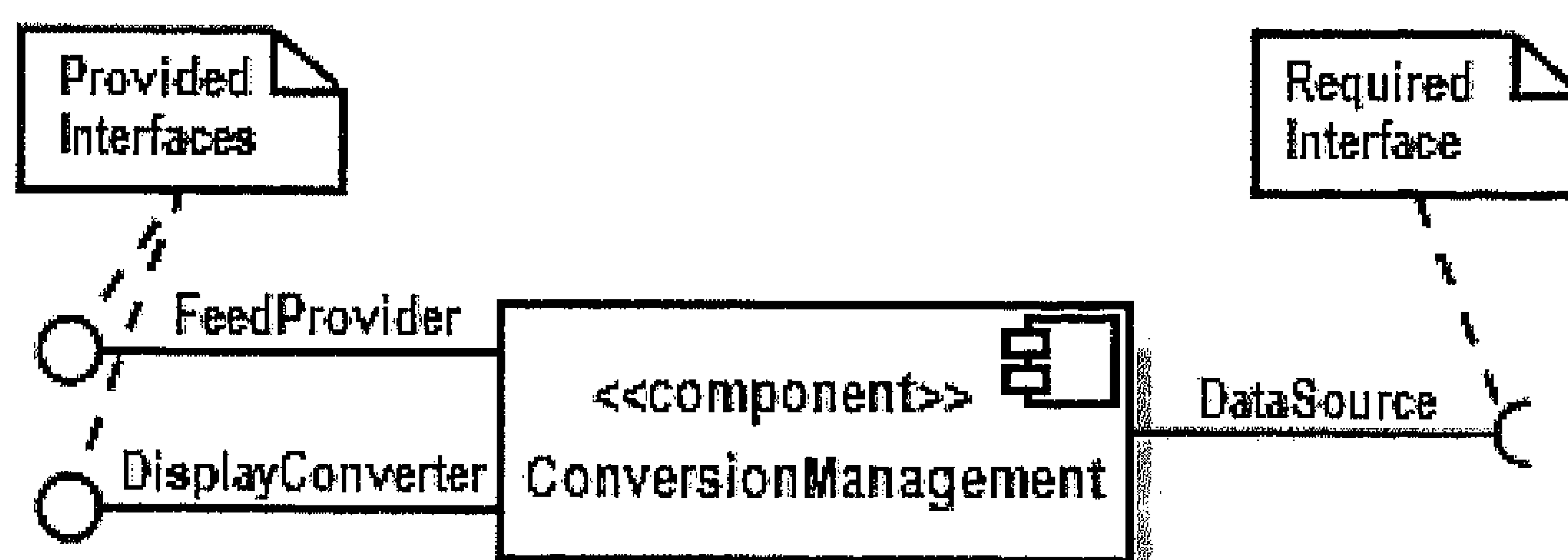
هناك ثلاثة أساليب نموذجية لعرض الواجهات المُتوفّرة والمُتطلّبة في لغة النمذجة الموحدة: أسلوب رمز الكرة ورمز المقبس، وأسلوب ترميز الحاشية، وأسلوب القوائم النصية.

١٢-٣-١ ترميز الكرة و المقبس للواجهات
Ball and Socket Notation for Interfaces

يمكن عرض الواجهة المُتوفّرة للمكوّن باستعمال رمز الكرة المقدم في الفصل الخامس. ويتم عرض الواجهة المُتطلّبة باستعمال الرمز النظير للكرة وهو رمز المقبس المرسوم كنصف دائرة ممتدة من خط محدد. ويجب كتابة اسم الواجهة بالقرب من الرموز التي تمثلها.

ويعرض الشكل رقم (١٢-٤) أن المكوّن ConversionManagement يوفر الواجهتين FeedProvider و DisplayConverter ويتطلب الواجهة DataSource.

ويعتبر ترميز الكرة والمقبس الوسيلة الأكثر شيوعاً لعرض واجهات المكوّن مقارنة بالأسلوبين التاليين.

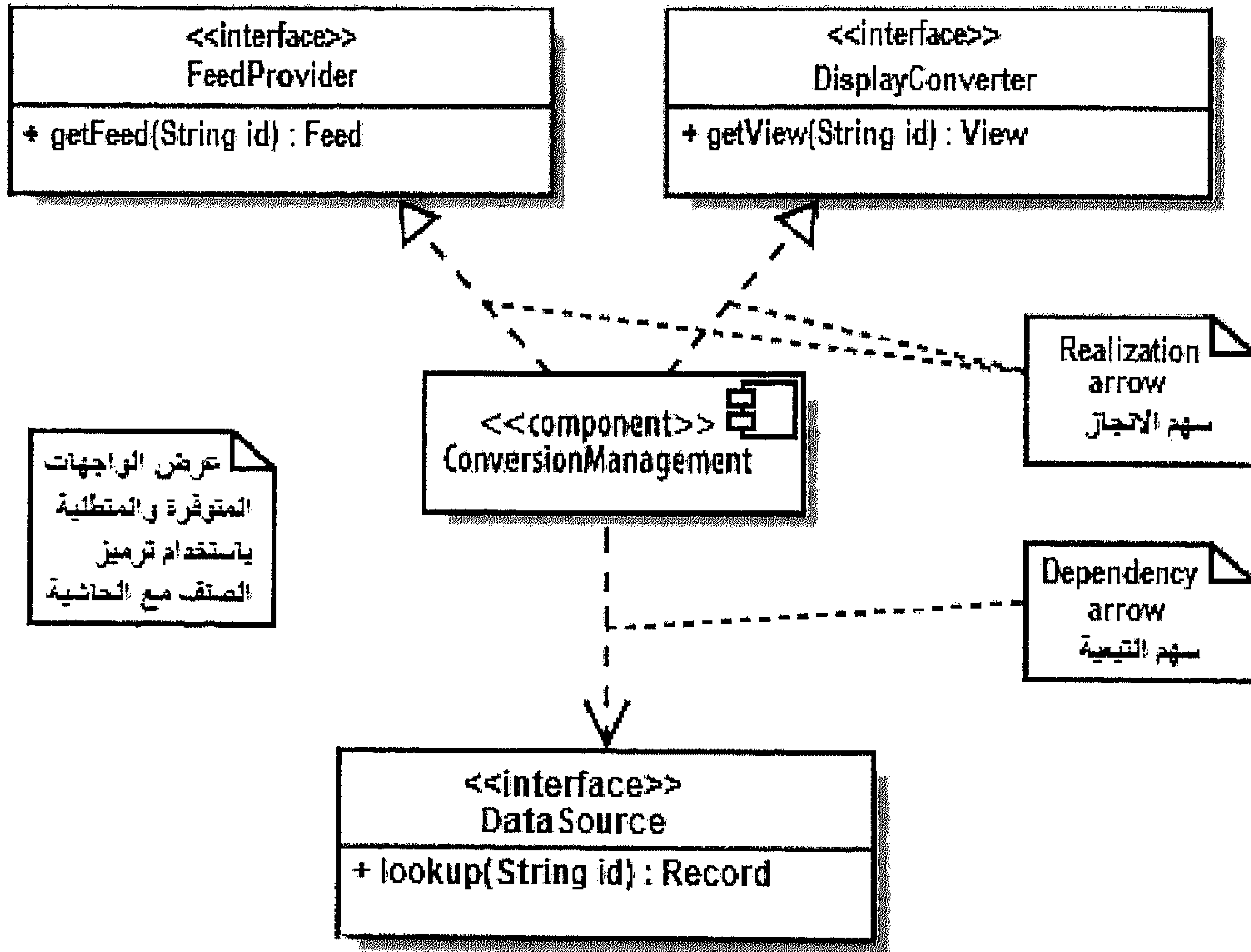


شكل رقم (١٢-٤) ترميز الكرة و المقبس لعرض الواجهات المُتوفّرة و المُتطلّبة للمكوّن.

١٢-٣-٢ ترميز الحاشية للواجهات

Stereotype Notation for Interfaces

يمكن أيضاً عرض الواجهات المُتطلَّبة و المُتوفّرة للمكوّن من خلال رسم الواجهات باستعمال ترميز الصنف مع حاشية (المقدم في الفصل الخامس). إذا كان المكوّن ينجز واجهة محددة، فارسم سهم الإنجاز realization من المكوّن إلى الواجهة. وإذا كان المكوّن يتطلب واجهة ما، فارسم سهم الاعتمادية dependency من المكوّن إلى الواجهة، كما هو معروض في الشكل رقم (١٢-٥).

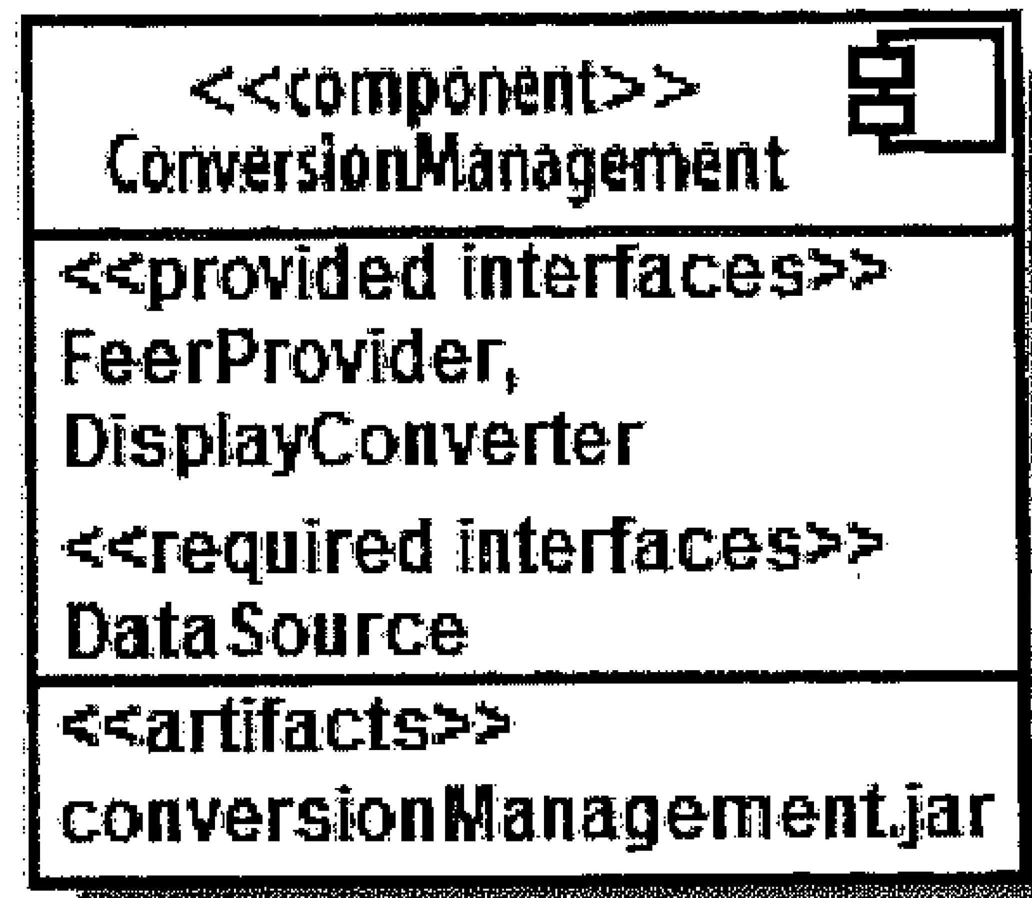


شكل رقم (١٢-٥) عرض عمليات الواجهات المُتطلَّبة و المُتوفّرة باستعمال ترميز الصنف مع حاشية.

يفيد هذا الترميز إذا أردت عرض عمليات الواجهات. وإذا لم ترغب بذلك، فيكون من الأفضل استعمال ترميز الكرة والمقبس، لأنه يعرض نفس المعلومات بشكل متراص ومحكم أكثر.

١٢-٣-٣ قوائم واجهات المكوّن Listing Component Interfaces

إن الأسلوب الأكثر تراصاً لعرض الواجهات المُتطلّبة و المُتوفّرة هو بتسجيلهم داخل المكوّن، حيث يتم ذلك بشكل منفصل، كما هو معروض في الشكل رقم (١٢-٦).



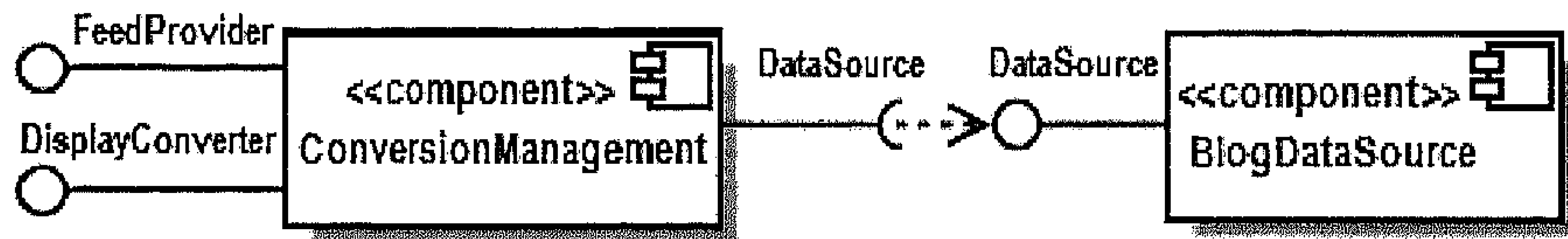
شكل رقم (١٢-٦) يعتبر تسجيل الواجهات المُتطلّبة و المُتوفّرة داخل المكون التمثيل الأكثر تراصاً.

يوجد القسم الإضافي <<artifacts>> مع هذا الترميز، والذي يقوم بتسجيل الأدوات المصنّعة artifacts، أو الملفات الفيزيائية، التي يكشف عنها المكوّن. وبما أن الأدوات المصنّعة تهتم بكيفية نشر النظام، فقد تمت مناقشتها في مخططات النشر (انظر إلى الفصل الخامس عشر). يعتبر تسجيل الأدوات المصنّعة داخل المكوّن أسلوباً بديلاً للأساليب الخاصة بعرض الأدوات المصنّعة التي تكشف عنها المكوّنات (معروضة في الفصل الخامس عشر).

ويعتمد اختيار الترميز الذي يجب استعماله للواجهات المُتطلّبة و المُتوفّرة على ما تحاول أن توصله من خلاله. و يمكن الإجابة بشكل جيد عن هذا السؤال عند درس المكوّنات و هي تعمل معاً.

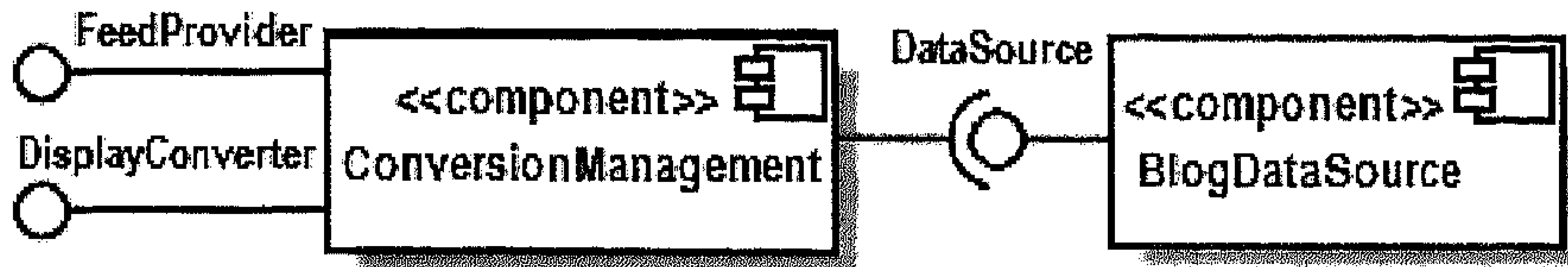
١٢-٤ عرض المكوّنات التي تعمل معاً
Showing Components Working Together

إذا كان للمكوّن واجهة مُتطلّبة، بالتالي هو يحتاج إلى صنف أو مكوّن آخر في النظام يقوم بتوفير هذه الواجهة. لإظهار اعتماد مكوّن له واجهة مُتطلّبة على مكوّن آخر يقوم بتوفيرها، ونقوم برسم سهم اعتمادية من رمز مقبس المكوّن المُعتمِد إلى رمز كرة المكوّن المُوفّر، كما هو معروض في الشكل رقم (١٢-٧).



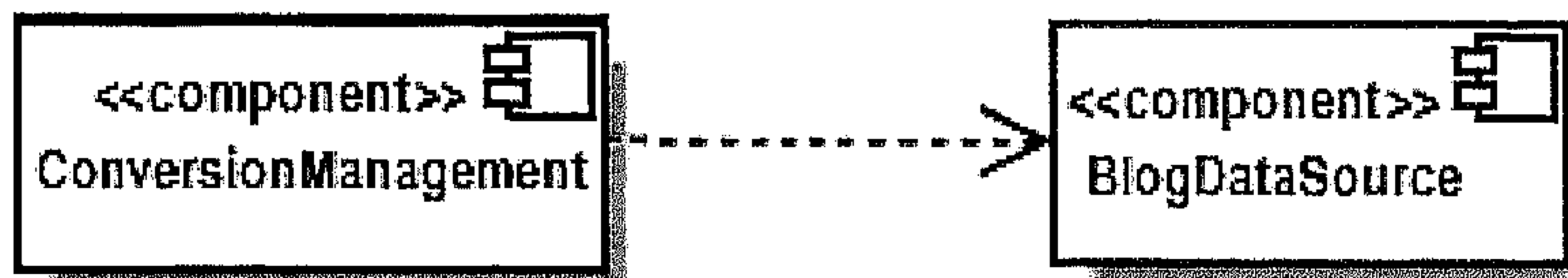
شكل رقم (١٢-٧) يتطلّب المكوّن ConversionManagement الواجهة DataSource، ويُوفر المكوّن BlogDataSource تلك الواجهة.

قد تسمح لنا أداة لغة النمذجة الموحدة المستعملة بتقديم عرض اختياري للشكل رقم (١٢-٧)، وذلك بتحريك الكرة والمقبس معاً ليصبحا متلاصقَين (حذف سهم الاعتمادية)، كما هو معروض في الشكل رقم (١٢-٨). إنه ترميز رابط التجميع بالحقيقة الذي سيقدم لاحقاً في هذا الفصل.



شكل رقم (١٢-٨) عرض اختياري يقوم بلصق الكرة بالمقبس.

يمكن أيضاً حذف الواجهة ورسم علاقة الاعتمادية مباشرة بين المكوّنات، كما هو معروض في الشكل رقم (١٢-٩).



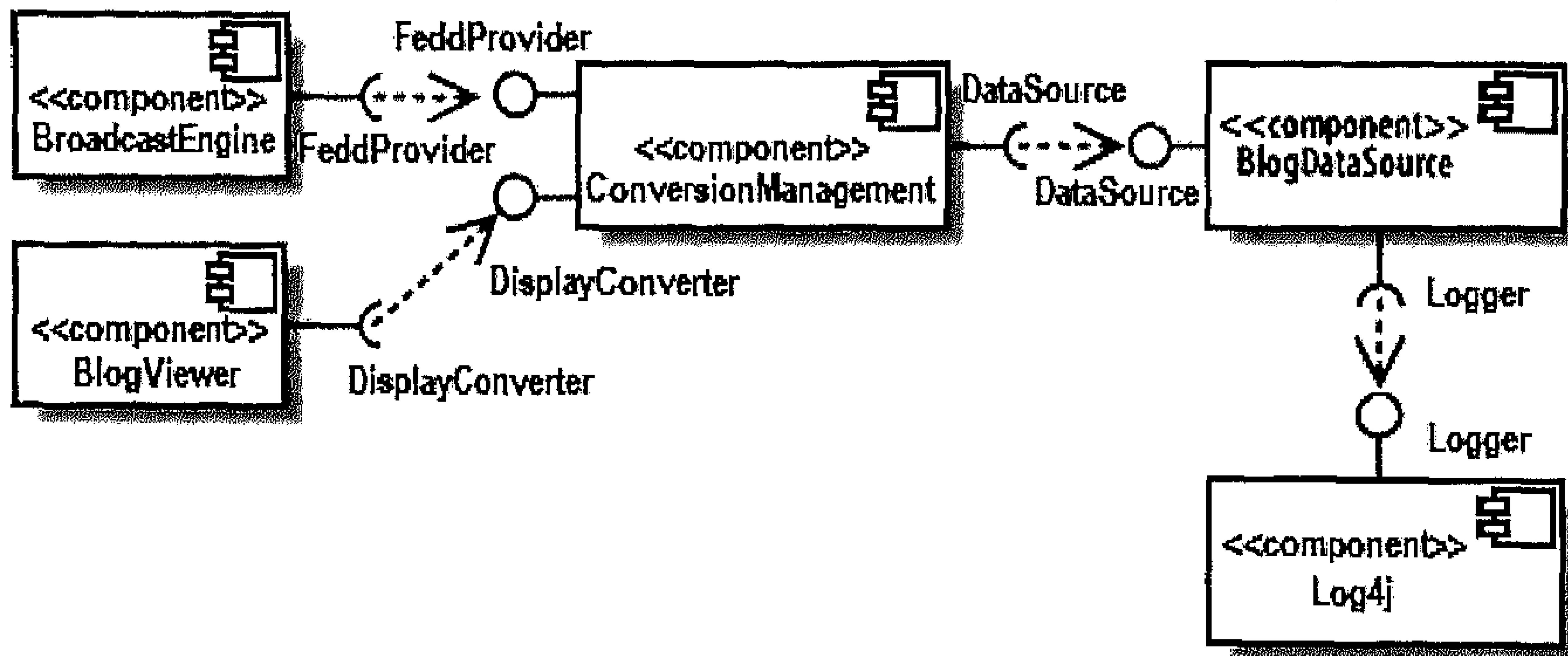
شكل رقم (١٢-٩) يمكن رسم أسهم الاعتمادية مباشرة بين المكوّنات وذلك لعرض رؤية عالية المستوى.

إن الترميز الذي لا يظهر الواجهة (معروض في الشكل رقم (١٢-٩) أبسط من الترميز الذي يظهر الواجهة (معروض في الشكل رقم (١٢-٧)، لذلك قد تجرب استعمال هذا الترميز للاختزال، لكن عليك أن تتذكر بضعة عوامل عند اختيار كيفية رسم اعتماديات المكوّنات.

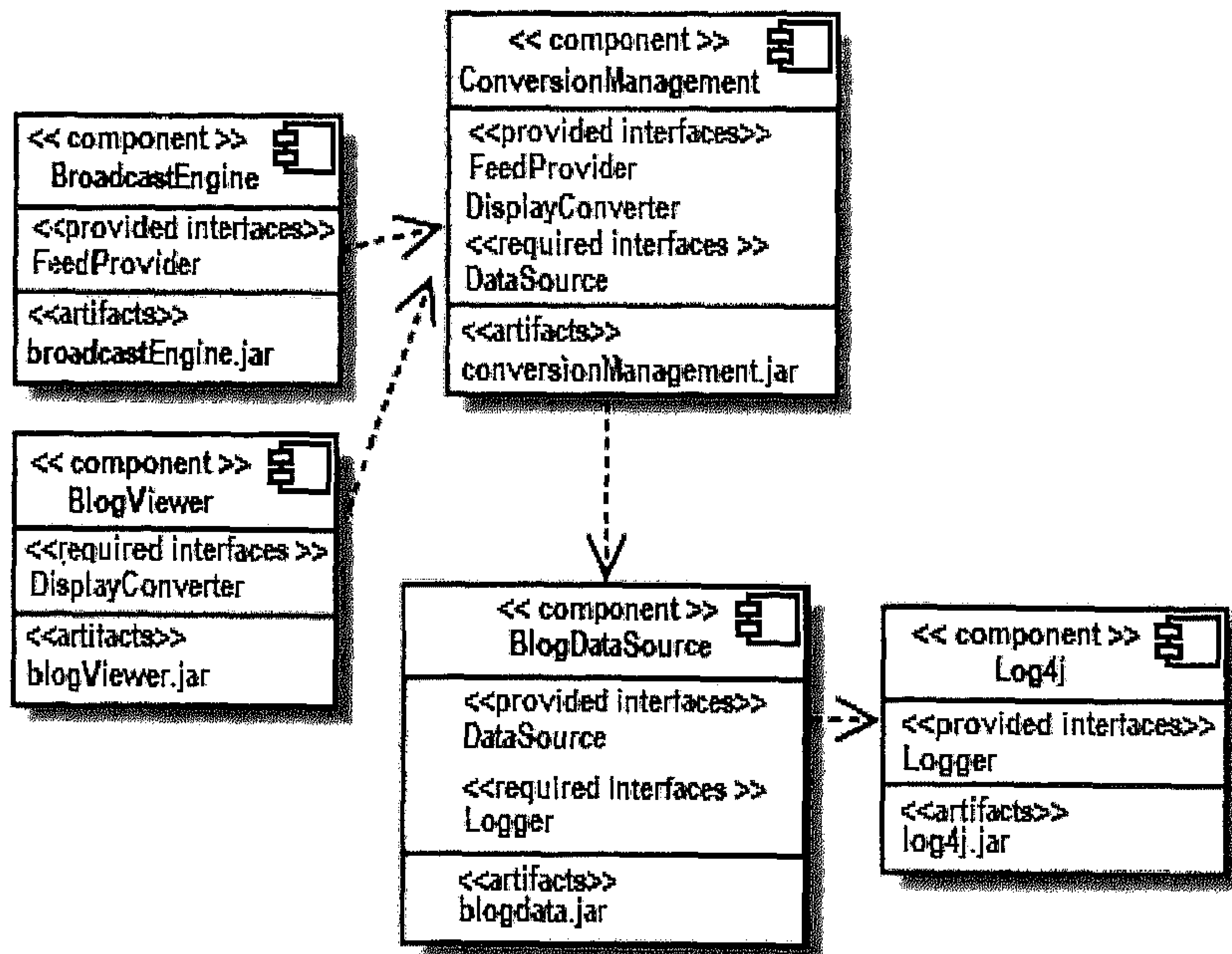
تذكر أنّ الواجهات تساعد على بقاء المكوّنات مقترنة بشكل ضعيف، لذلك فهي تعتبر عاملاً مهماً في معمارية المكوّنات. إن عرض المكوّنات الرئيسية في النظام وترابطهم فيما بينهم من خلال الواجهات هو أسلوب مهم لوصف معمارية النظام، وهذا ما يجيد عمله الترميز الذي يلصق الكرة بالمقبس، كما هو معروض في الشكل رقم (١٢-١٠).

ويعتبر الترميز الذي لا يظهر الواجهات جيداً في عرض رؤيات عالية المستوى ومبسّطة لاعتماديات المكوّنات. قد يفيد ذلك في فهم إدارة

ترتيبات النظام أو اهتمامات النشر، لأن التركيز على اعتماديات المكوّنات وتسجيل الأدوات المصنّعة الظاهرة، يسمح برؤية المكوّنات والملفات ذات العلاقة التي تكون متطلبة أثناء النشر، كما هو معروض في الشكل رقم (١٢-١١).



شكل رقم (١٢-١٠) التركيز على المكوّنات و الواجهات الرئيسية في النظام.

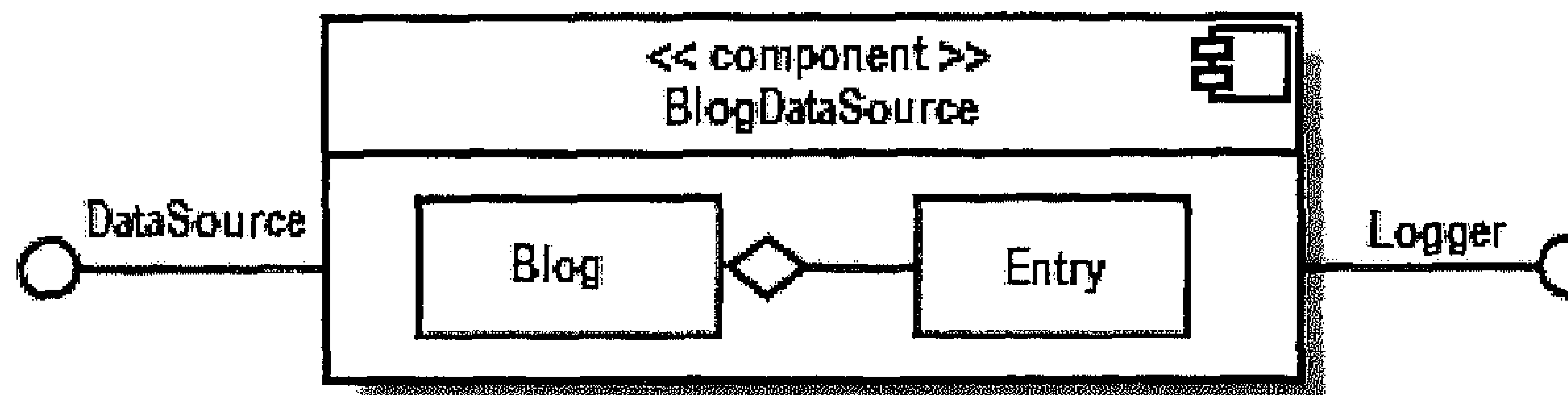


شكل رقم (١٢-١١) يفيد التركيز على اعتماديات المكوّن والأدوات المصنّعة الظاهرة عند محاولة التحكم بترتيبات النظام ونشره.

١٢-٥ الأصناف المُنجِزة للمكوّن
Classes That Realize a Component

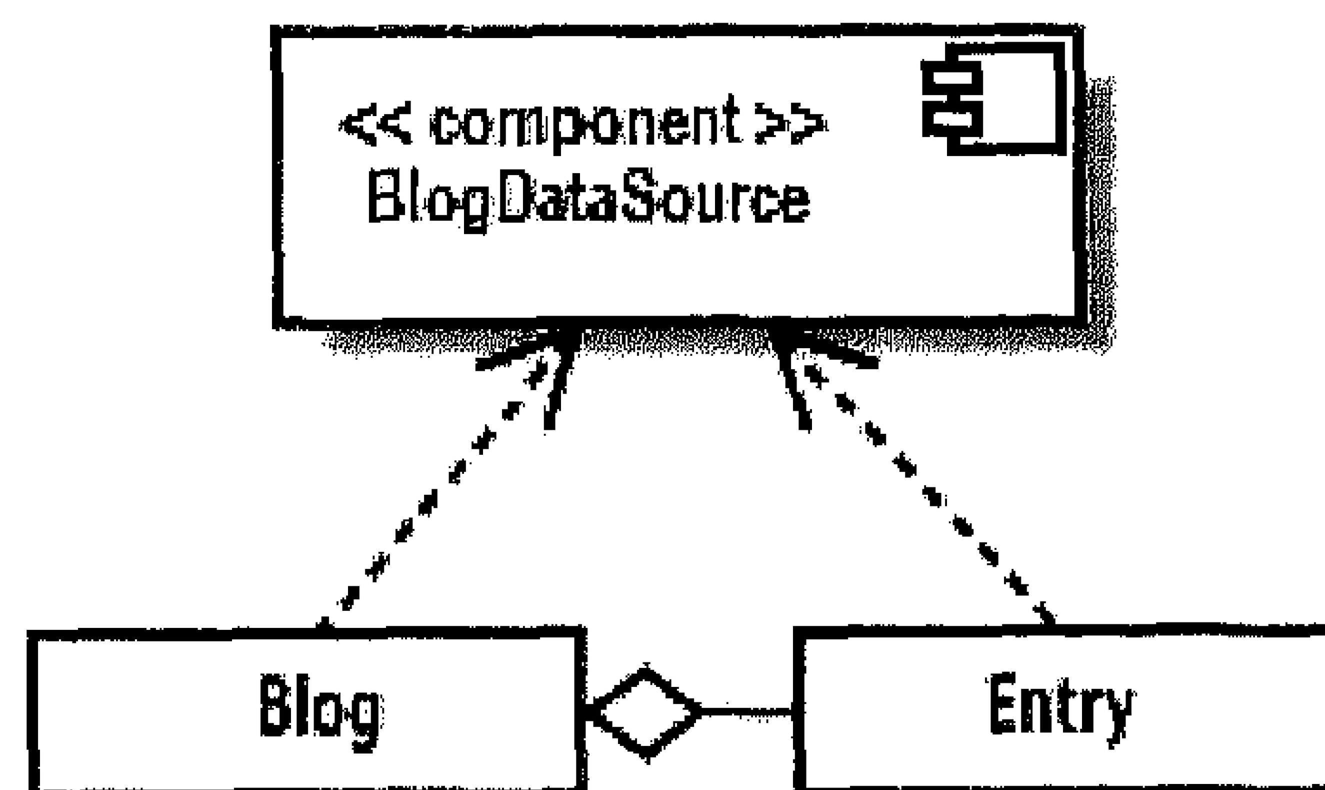
عادة ما يحتوي و يستعمل المكوّن أصنافاً أخرى لإنجاز و ظيفته. ويقال لتلك الأصناف أنها تُنجِز **realize** مكوّنا محدداً، فهي تساعد المكوّن في إنجاز عمله.

ويمكن عرض الأصناف المُنجِزة برسمهم (وعلاقاتهم) داخل المكوّن. ويبين الشكل رقم (١٢-١٢) أن المكوّن BlogDataSource يحتوي على الصنفَين مدونة **Blog** وتدوينة **Entry**، ويعرض أيضاً علاقة التجميع التي بينهما.



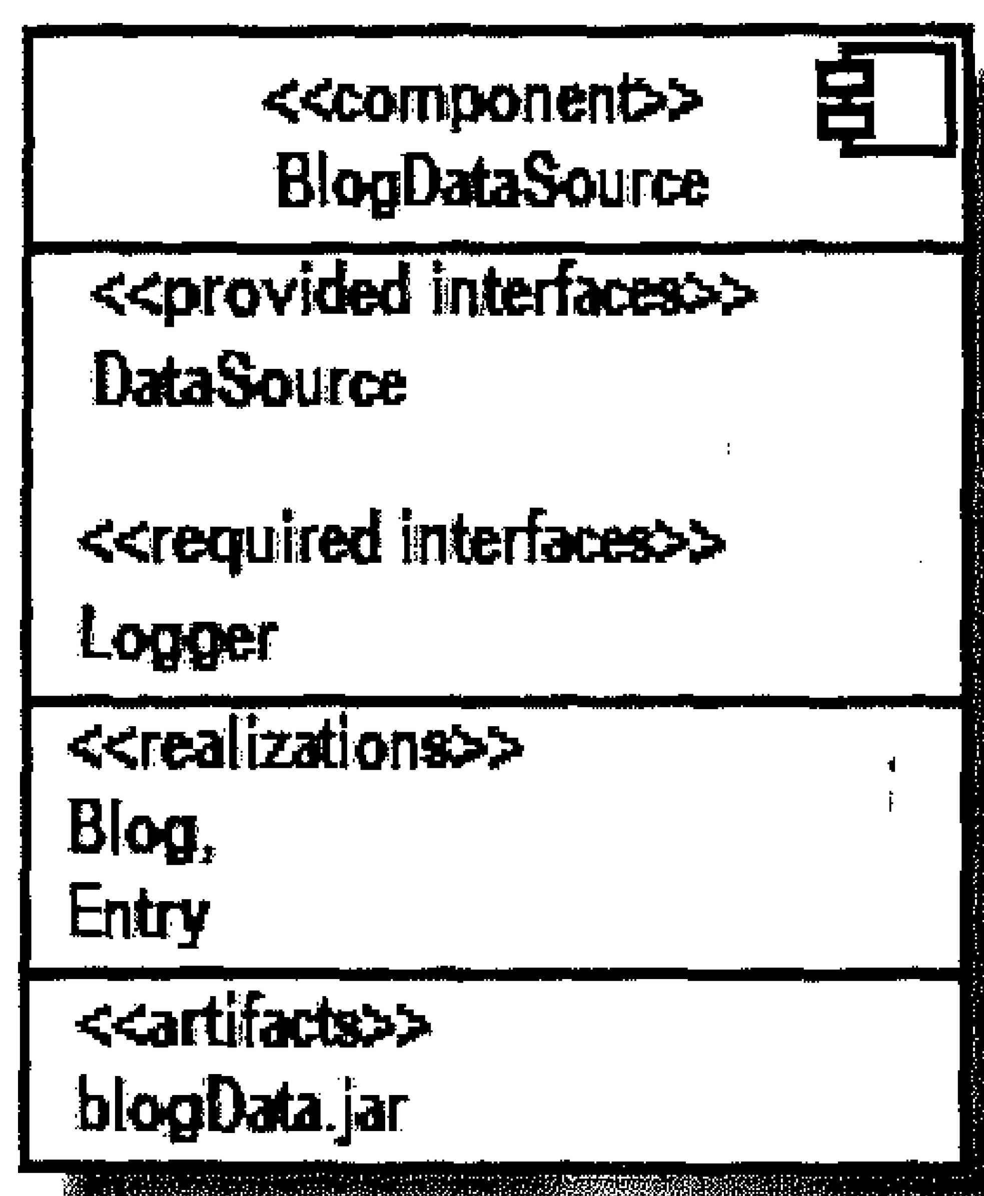
شكل رقم (١٢-١٢) ينجز الصنفان **Blog** و **Entry** المكوّن **BlogDataSource**.

يمكن أيضاً عرض الأصناف المُنجِزة لمكوّن ما برسمهم خارج المكوّن مع سهم اعتمادية من الصنف المُنجِز إلى المكوّن، كما هو معروض في الشكل رقم (١٢-١٣).



شكل رقم (١٢-١٣) رؤية بديلة تعرض الأصناف المُنجِزة خارج المكوّن وعلاقة الاعتمادية التي بينهما.

الأسلوب الأخير لعرض الأصناف المُنجزة هو بتسجيلهم في مقصورة الإنجازات <<realizations>> داخل المكوّن، كما هو معروض في الشكل رقم (١٢-١٤).

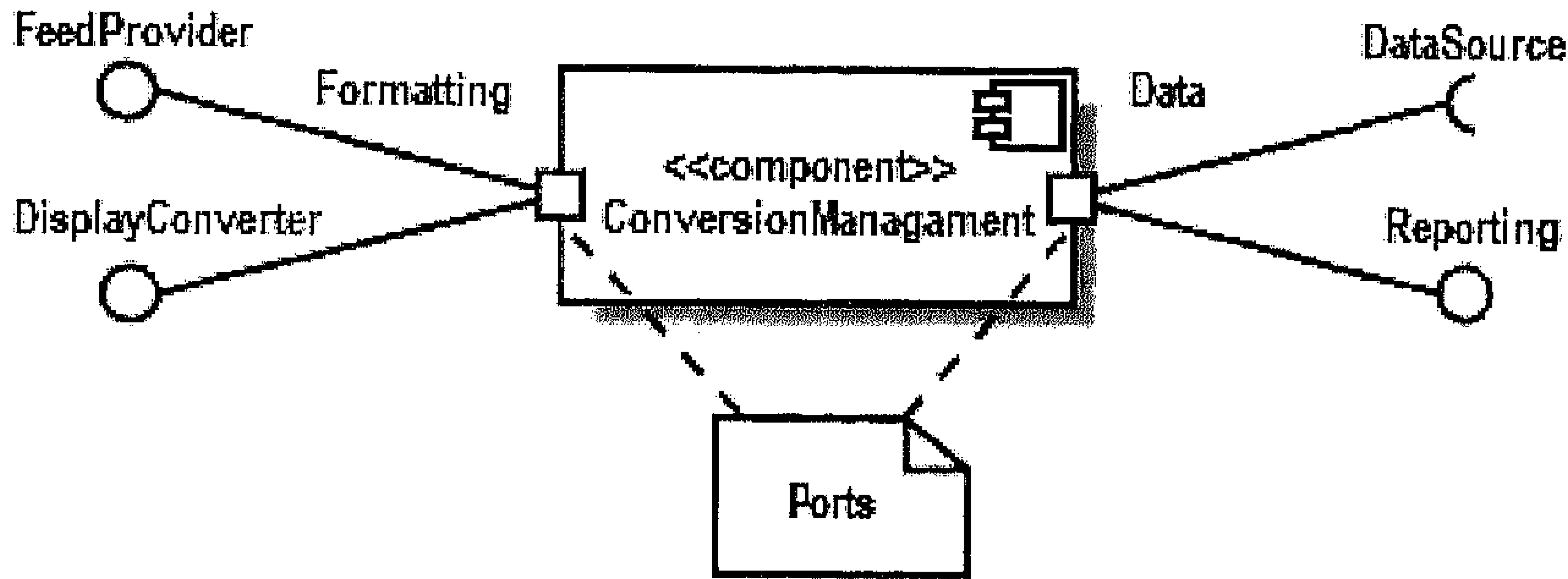


شكل رقم (١٢-١٤) تسجيل الأصناف المُنجزة داخل المكوّن.

كيف تقرر أي ترميز عليك استعماله لعرض الأصناف المُنجزة للمكوّن؟ ربما تكون محدوداً بأداة لغة النمذجة الموحدة المستعملة، لكن إذا كان لديك الخيار، يفضل الكثير من المنمذجين الترميز الأول (رسم الأصناف المُنجزة داخل المكوّن بدلاً من رسمهم خارج المكوّن)؛ لأن رسمهم بالداخل يؤكد بشكل مرئي أنّ تلك الأصناف تبني المكوّن لإنجاز وظيفته. قد يفيد تسجيل الأصناف المُنجزة إذا كنت تريد شيئاً متراصاً، لكن تذكر أنه لا يمكن عرض العلاقات التي بين الأصناف المُنجزة، بينما يمكن أن يقوم أول ترميزين بذلك.

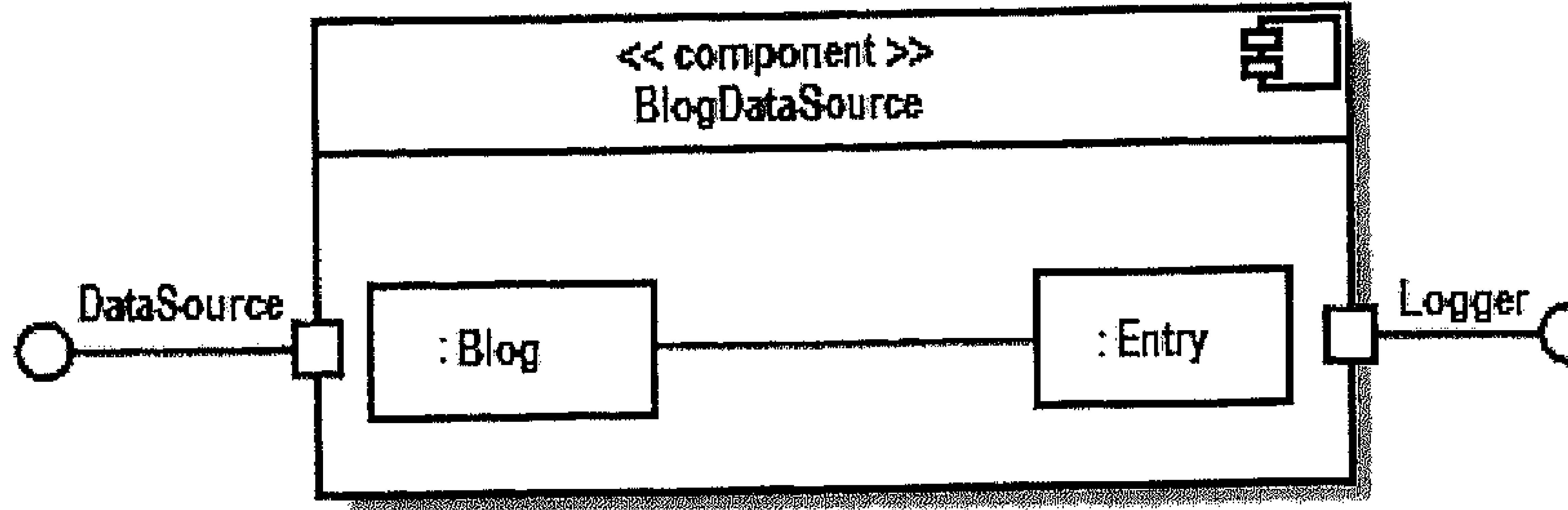
١٢-٦ المنافذ و الهيكل الداخلي
Ports and Internal Structure

لقد تم تقديم المنافذ والهيكل الداخلي للصنف في الفصل الحادي عشر. ويمكن أيضاً أن يكون للمكوّنات منافذ وهيكل داخلي. ويمكن استعمال المنافذ لنمذجة الأساليب المختلفة التي يمكن أن يستعملها المكوّن مع الواجهات ذات العلاقة والتي يتم ربطها بالمنفذ. ويبين الشكل رقم (١٢-١٥)، أن للمكوّن ConversionManagement المنفذ تهيئة **Formating** و المنفذ بيانات **Data**، و كلاً منهما موصول بواجهاته المرتبطة به.



شكل رقم (١٢-١٥) تعرض المنافذ الاستعمالات الفريدة للمكوّن وتجمع "هكذا" واجهات.

يمكن إظهار الهيكل الداخلي للمكوّن من أجل نمذجة أجزائه وميزاته وروابطه (انظر الفصل الحادي عشر لمراجعة الهيكل الداخلي). يعرض الشكل رقم (١٢-١٦) الهيكل الداخلي للمكوّن BlogDataSource.



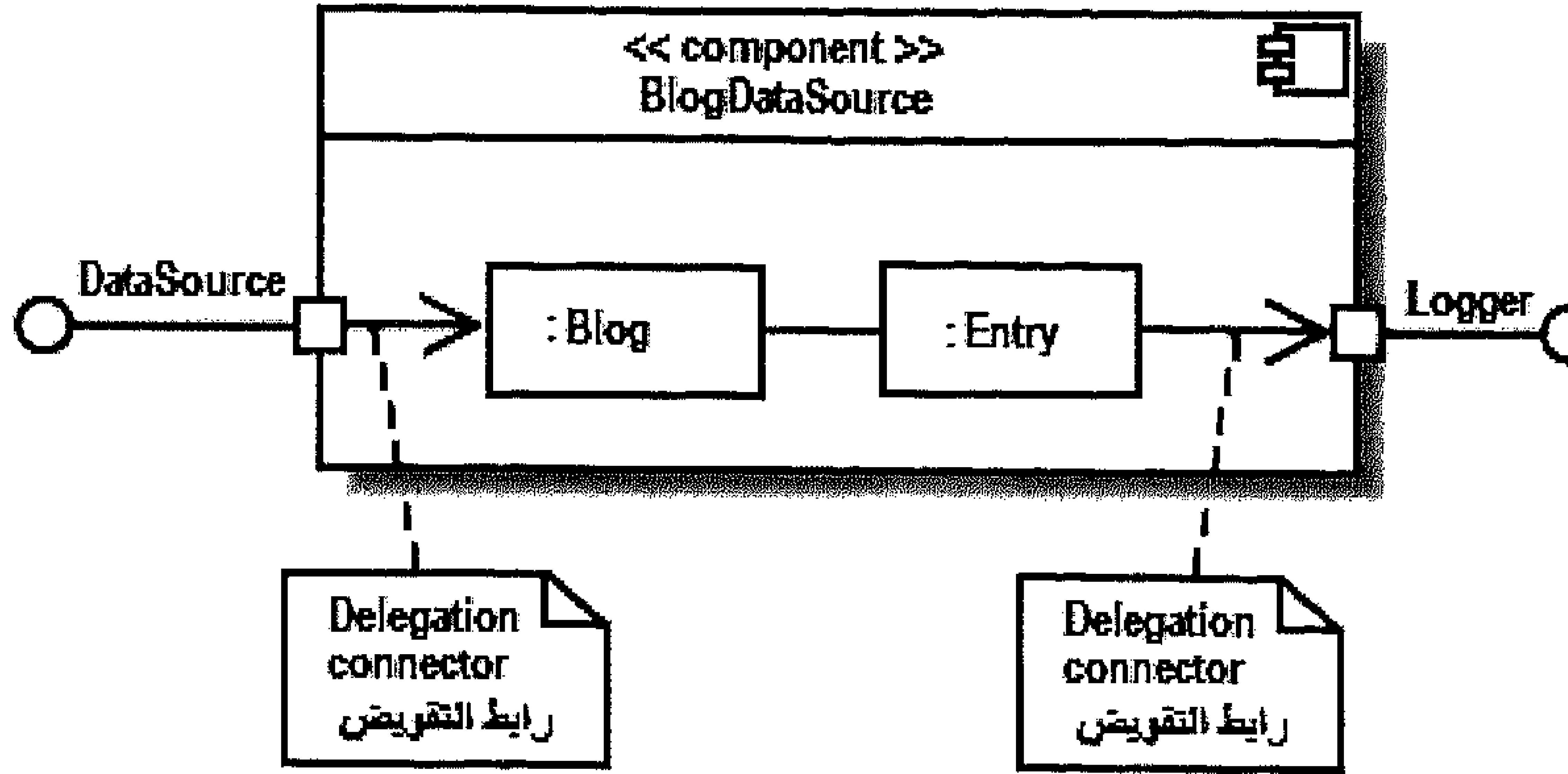
شكل رقم (١٢-١٦) الهيكل الداخلي للمكوّن **BlogDataSource**.

إن للمكوّنات مفاهيم فريدة خاصة بها عند عرض المنافذ والهيكل الداخلي، وتسمى روابط التفويض وروابط التجميع، ويتم استعمالها لعرض كيف تتماشى واجهات المكوّن مع أجزائه الداخلية وكيف تعمل الأجزاء الداخلية معاً.

١٢-٦-١ روابط التفويض Delegation Connectors

يمكن إنجاز واجهة مُتوفّرة للمكوّن بواسطة أحد أجزائه الداخلية. بشكل مماثل يمكن أن تكون واجهة مُتطلّبة للمكوّن مطلوبة من قِبَل إحدى أجزائه. ويمكن في هذه الحالات استعمال روابط التفويض لإظهار أن الأجزاء الداخلية تُنجز أو تستعمل واجهات المكوّن.

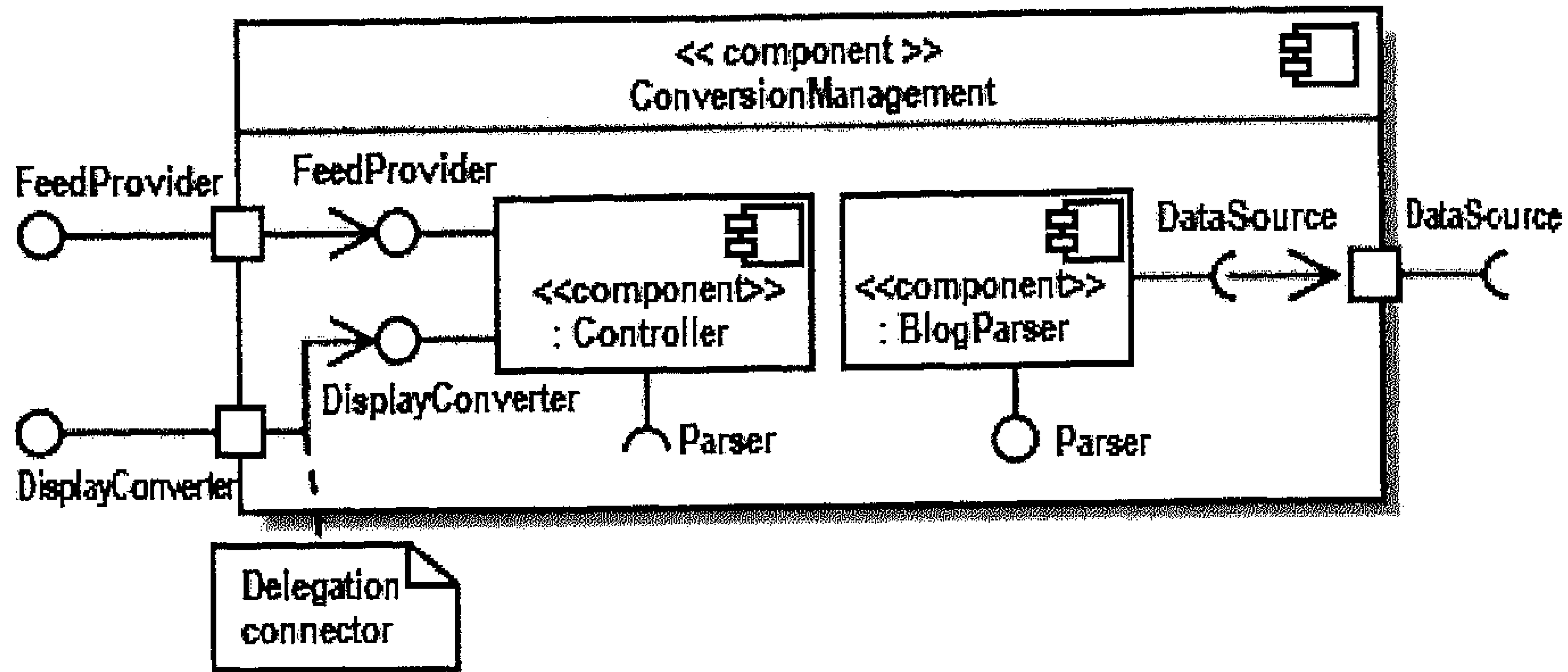
ويتم رسم روابط التفويض باستعمال أسهم تشير إلى "اتجاه حركة المرور"، تربط المنفذ المتصل بالواجهة مع الجزء الداخلي. وإذا أنجز الجزء الداخلي واجهة مُتوفّرة، عندئذ يشير السهم من المنفذ إلى الجزء الداخلي. وإذا استعمل الجزء الداخلي واجهة مُتطلّبة، عندئذ يشير السهم من الجزء الداخلي إلى المنفذ. يعرض الشكل رقم (١٢-١٧) مثالاً لاستعمال روابط التفويض لربط الواجهات مع الأجزاء الداخلية.



شكل رقم (١٢-١٧) تبين روابط التفويض كيف تتقابل الواجهات و الأجزاء الداخلية: ينجز الصنف **Blog** الواجهة **DataSource** و يتطلب الصنف **Entry** الواجهة **Logger**.

يمكن تخيّل روابط التفويض كالتالي: يمثل المنفذ فتحة في المكوّن تمر عبرها الاتصالات، وتشير روابط التفويض إلى اتجاه الاتصال. لذا، يمثل رابط التفويض الذي يتجه من منفذ ما إلى جزء داخلي الرسائل الممررة للجزء الذي سيقوم بمعالجتها.

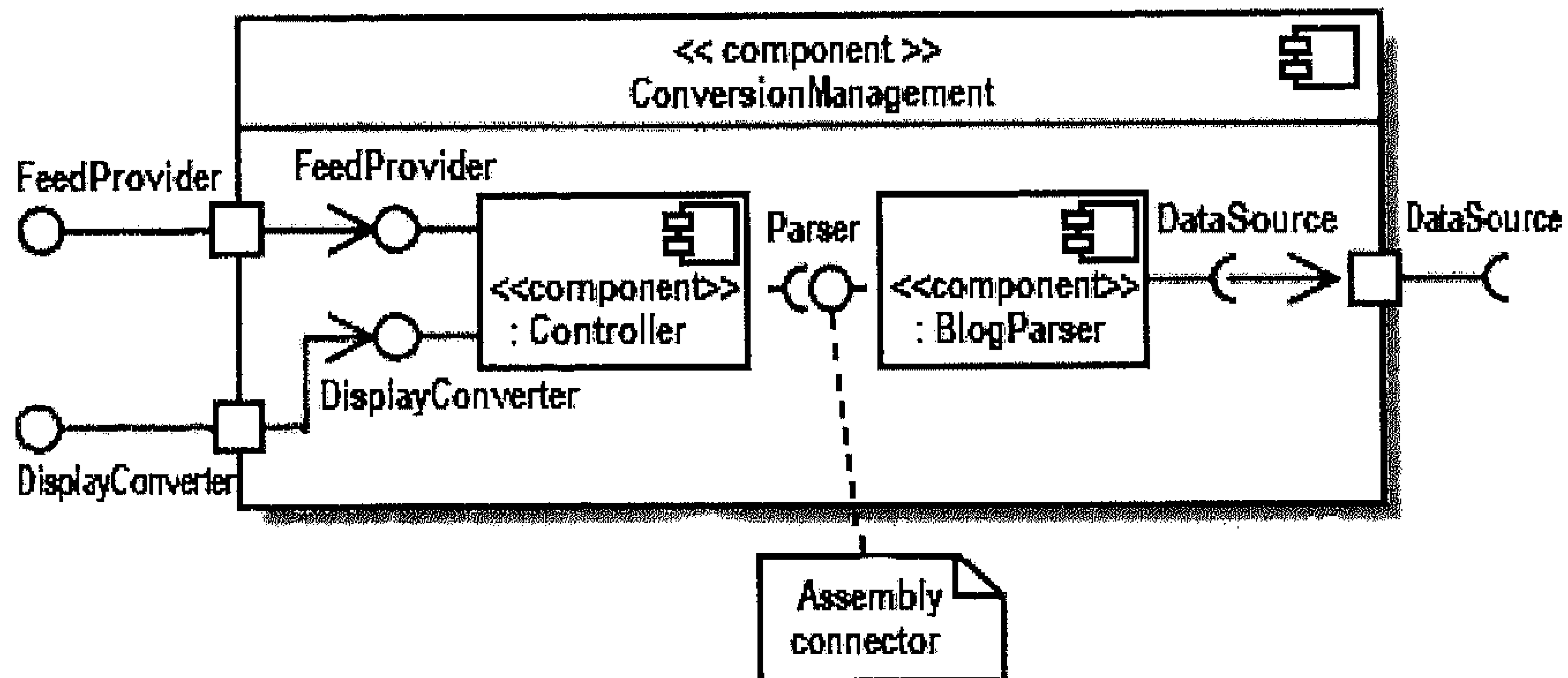
وعند عرض واجهات الأجزاء الداخلية، يمكن توصيل روابط التفويض بالواجهة بدلاً من توصيلها مباشرة بالجزء الداخلي. يستعمل هذا بشكل شائع عند عرض مكوّن يحتوي على مكوّنات أخرى. ويعرض الشكل رقم (١٢- ١٨) مثالاً عن هذا الترميز. يحتوي المكوّن ConversionManagement على المكوّنَين Controller و BlogParser. يوفّر المكوّن ConversionManagement الواجهة FeedProvider، لكن يتم فعلياً إنجاز ذلك داخلياً باستعمال الجزء الداخلي Controller.



شكل رقم (١٢-١٨) تستطيع روابط التفويض أيضاً توصيل واجهات الأجزاء الداخلية مع المنافذ.

١٢-٦-٢ روابط التجميع Assembly Connectors

تبين روابط التجميع أن المكوّن يتطلب واجهة يوفرها مكوّن آخر. تقوم روابط التجميع بلصق رمز الكرة مع رمز المقبس الممثلان للواجهات المُتطلّبة والمُتوفّرة. يعرض الشكل رقم (١٢-١٩) ترميز رابط التجميع وهو يوصل المكوّن Controller بالمكوّن BlogParser.



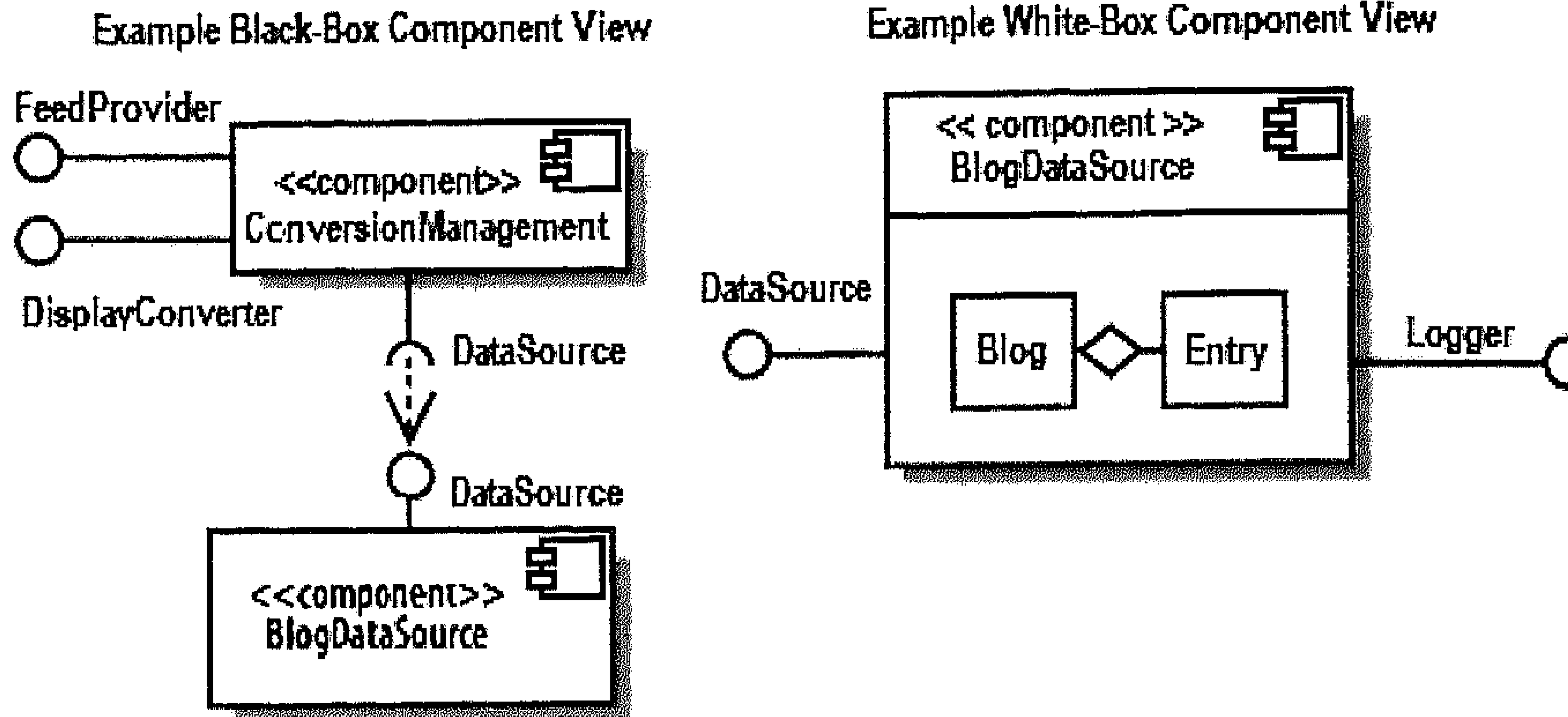
شكل رقم (١٢-١٩) تعرض روابط التجميع المكوّنات التي تعمل معاً من خلال الواجهات.

وتشكل روابط التجميع أنواعاً خاصّة من الروابط التي تم تعريفها للاستعمال عند إظهار الهياكل المركبة للمكوّنات. لاحظ أن المكوئين Controller وBlogParser يستعملان الترميز roleName:className المقدّم في الهياكل المركبة، ويساعدان على تشكيل الهيكل الداخلي للمكوّن ConversionManagement. لكن يتم أيضاً استعمال روابط التجميع أحياناً كعرض اختياري لاعتمادية المكوّن من خلال الواجهات بشكل عام، كما هو معروض سابقاً في الشكل رقم (١٢-٨).

١٢-٧ منظورا الصندوق الأسود و الصندوق الأبيض للمكوّن
Black-Box and White-Box Component Views

هناك منظوران للمكوّنات في لغة النمذجة الموحدة: منظور الصندوق الأسود ومنظور الصندوق الأبيض. ويعرض منظور الصندوق الأسود مظهر المكوّن من الخارج، بما فيه واجهاته المُتطلّبة وواجهاته المُتوفّرة، كما يعرض كيفية تعلقه بالمكوّنات الأخرى. لا يحدد منظور الصندوق الأسود أي أمر خاص بالإنجاز الداخلي للمكوّن. من الناحية الأخرى، يعرض منظور الصندوق الأبيض الأصناف والواجهات والمكوّنات الأخرى التي تساعد المكوّن على إنجاز وظيفته.

لقد رأينا ما يعرضه منظور الصندوق الأسود ومنظور الصندوق الأبيض. فما هو الفرق بينهما إذا من الناحية العملية؟ يعرض منظور الصندوق الأبيض الأجزاء التي داخل المكوّن، بينما لا يقوم منظور الصندوق الأسود بذلك، كما هو معروض في الشكل رقم (١٢-٢٠).



شكل رقم (١٢-٢٠) يفيد منظور الصندوق الأسود للمكوّن في عرض الوصف العام للمكوّنات التي في النظام، بينما يركز منظور الصندوق الأبيض على الأعمال الداخلية للمكوّن.

عند نمذجة النظام، من الأفضل استعمال منظور الصندوق الأسود للتركيز على الاهتمامات المعمارية واسعة النطاق. تتميز منظورات الصندوق الأسود في عرض المكوّنات الرئيسية في النظام وكيفية ترابطها. من ناحية أخرى، وتفيد منظورات الصندوق الأبيض في عرض كيفية إنجاز المكوّن وظيفته من خلال الأصناف التي يستعملها.

وعادة ما تحتوي منظورات الصندوق الأسود على أكثر من مكوّن واحد، بينما من الشائع تركيز منظور الصندوق الأبيض على محتويات مكوّن واحد.

١٢-٨ ما هي الخطوة التالية؟

بما أنك أصبحت تعرف الآن كيف تنمذج مكوّنات نظامك، فربما تريد النظر إلى كيفية نشر مكوّناتك على الأجهزة في مخططات الانتشار. ستتم تغطية مخططات الانتشار في الفصل الخامس عشر.

وهنـاك تـداخل حـاد بـين بعـض المواضـيع في مخططـات المكوّنـات والهياكل المركبة. لقد تم تعريف قدرة امتلاك المنافذ والهيكل الداخلي من أجل الأصناف في الهياكل المركبة. تقوم المكوّنات بوراثة هذه القدرة مع تقديم بعض مزاياها الخاصة مثل روابط التفويض والتجميع. (ارجع إلى الفصل الحادي عشر لمراجعة المنافذ والهيكل الداخلي للصنف).

تنظيم النموذج: الحُزَم

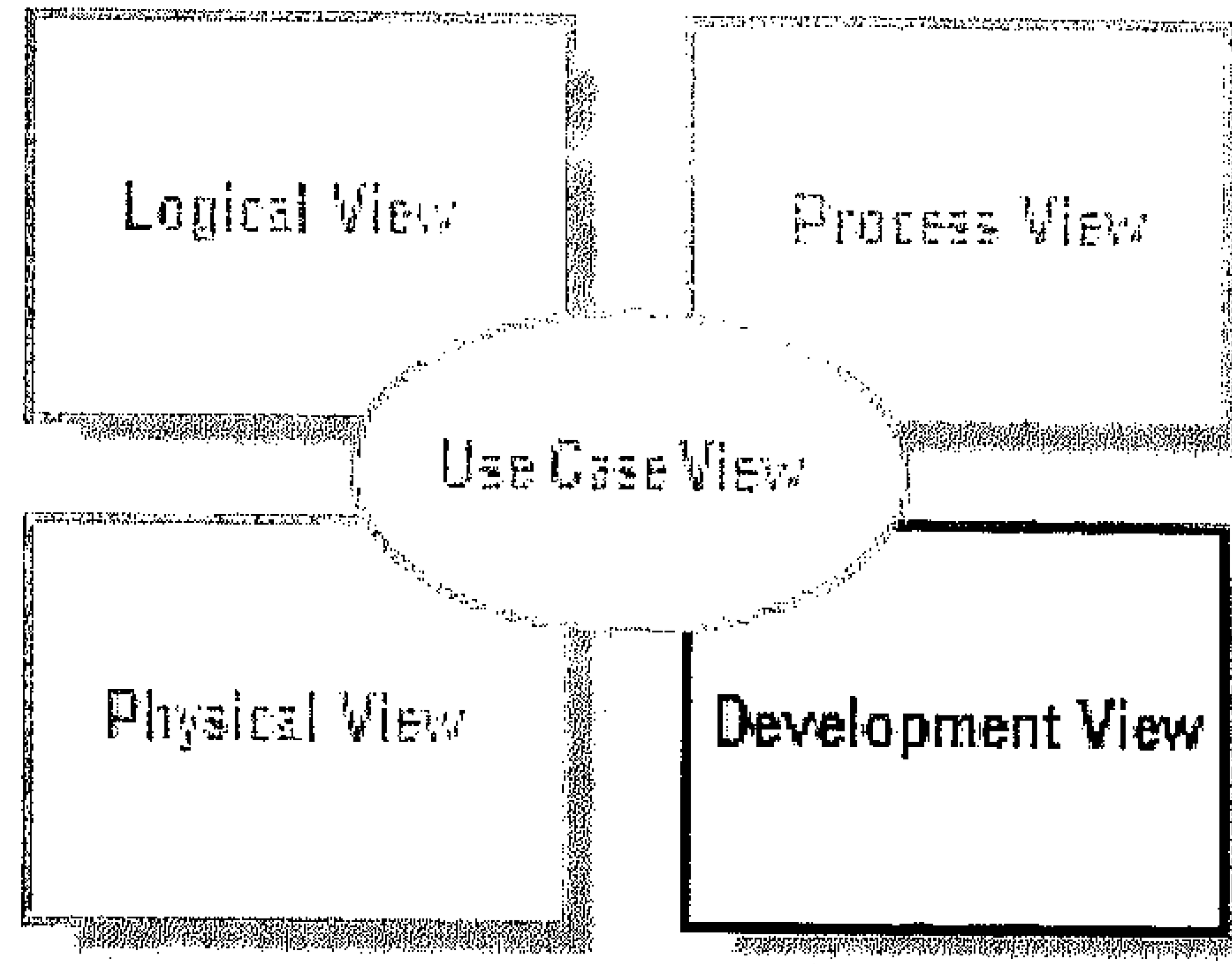
ORGANIZING YOUR MODEL: PACKAGES

بما أن البرمجيات تزداد في التعقيد، حيث بإمكانها احتواء مئات الأصناف بسهولة. إذا كنت مبرمجاً تعمل هكذا على مكاتب برمجية تتألف من الأصناف، فكيف ستتمكن من الإلمام بها؟ من أساليب فرض هيكلية لتنظيم الأصناف داخل مجموعات ذات علاقة منطقية. يمكن انتماء الأصناف المتعلقة بواجهة مستخدم التطبيق إلى مجموعة واحدة، كما يمكن انتماء الأصناف الخدماتية إلى مجموعة أخرى.

تتم نمذجة مجموعات الأصناف في لغة النمذجة الموحدة بواسطة الحُزَم. لدى معظم اللغات الكائنية التوجه عنصراً مماثلاً للحُزم في لغة النمذجة الموحدة، حيث يتم استعماله لتنظيم الأصناف وتجنّب التضارب بين أسمائها. على سبيل المثال، تضم لغة جافا الحُزَم، وتضم لغة C# فضاءات الأسماء **namespaces** (بالرغم من اختلاف الحُزَم بلغة جافا عن فضاءات الأسماء بلغة C# بشكل مهم في تفاصيل أخرى). ويمكن استعمال حُزَم لغة النمذجة الموحدة لنمذجة هذه الهيكليات.

وغالباً ما تستعمل مخططات الحُزم لمعاينة الاعتماديات بين الحُزَم. بما أن التغيير في حُزمة ما قد يتسبب بشرخ في حزمة أخرى معتمدة عليها، يعتبر فهم الاعتمادات بين الحُزَم أمر ضروري لاستقرار البرمجيات.

باستطاعة الحُزَم تنظيم أي عنصر من لغة النمذجة الموحدة تقريباً (لا تقتصر على تنظيم الأصناف فقط). على سبيل المثال، عادة ما تستعمل الحُزَم أيضاً لتجميع حالات الاستخدام. تشكّل مخططات الحُزم جزءاً من منظور التطوير الذي يهتم بكيفية تنظيم أجزاء النظام إلى وحدات وحُزَم، كما هو معروض في الشكل رقم (١٣-١).



شكل رقم (١٣-١) يصف منظور التطوير كيفية تنظيم أجزاء النظام إلى وحدات مستخدمة كحُزَم في لغة النمذجة الموحدة.

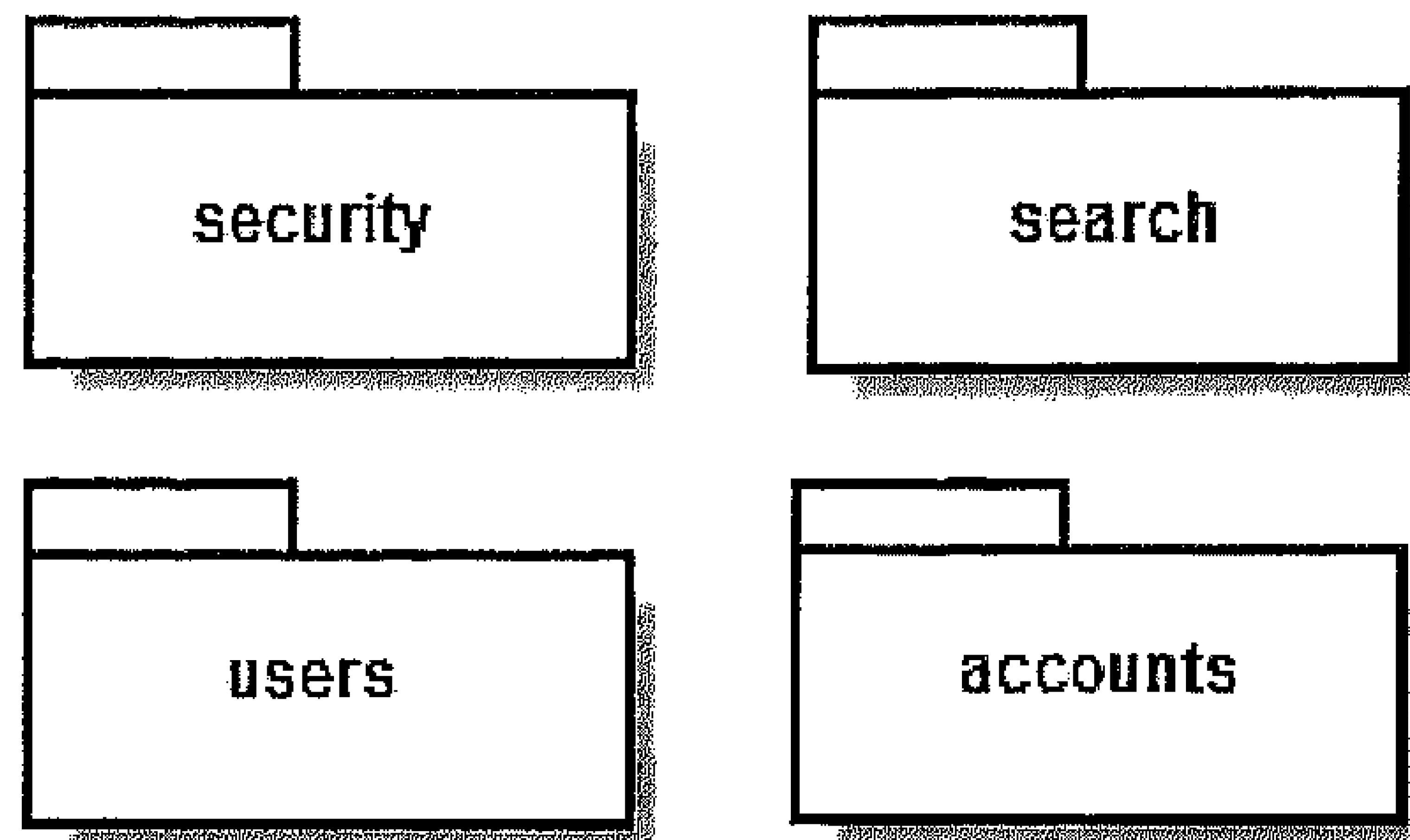
البداية Getting Started

يمكن ألا يكون لأداة لغة النمذجة الموحدة المستعملة مخطط حُزَم. تشكل الحُزَم هياكل تجميع يتم استعمالها لتنظيم أي عنصر في لغة النمذجة الموحدة تقريباً، لكن يشكل تنظيم الأصناف في مخططات الأصناف الاستعمال الأكثر شيوعاً لها. وتركز معظم أمثلة هذا الفصل على تطبيقات الحُزَم على الأصناف، لذلك قم بإنشاء مخطط أصناف جديد للعمل معاً على هذه الأمثلة.

١٣-١ الحُزَم Packages

افترض أنه أثناء تصميم نظام إدارة محتوى CMS، قد قرّرت الإبقاء على الأصناف المتعلقة بالأمن security مُجمّعة معاً (مثل إجراء

التثبّت من أصالة المستخدم (user authentication). يعرض الشكل رقم (١٣-٢) حُزمة الأمن وبعض الحُزَم الأخرى من نظام إدارة المحتوى في لغة النمذجة الموحدة. ويتم استعمال رمز حافظة الأوراق ذات العروة لتمثيل الحُزمة حيث يكتب اسم الحُزمة داخل الحافظة.

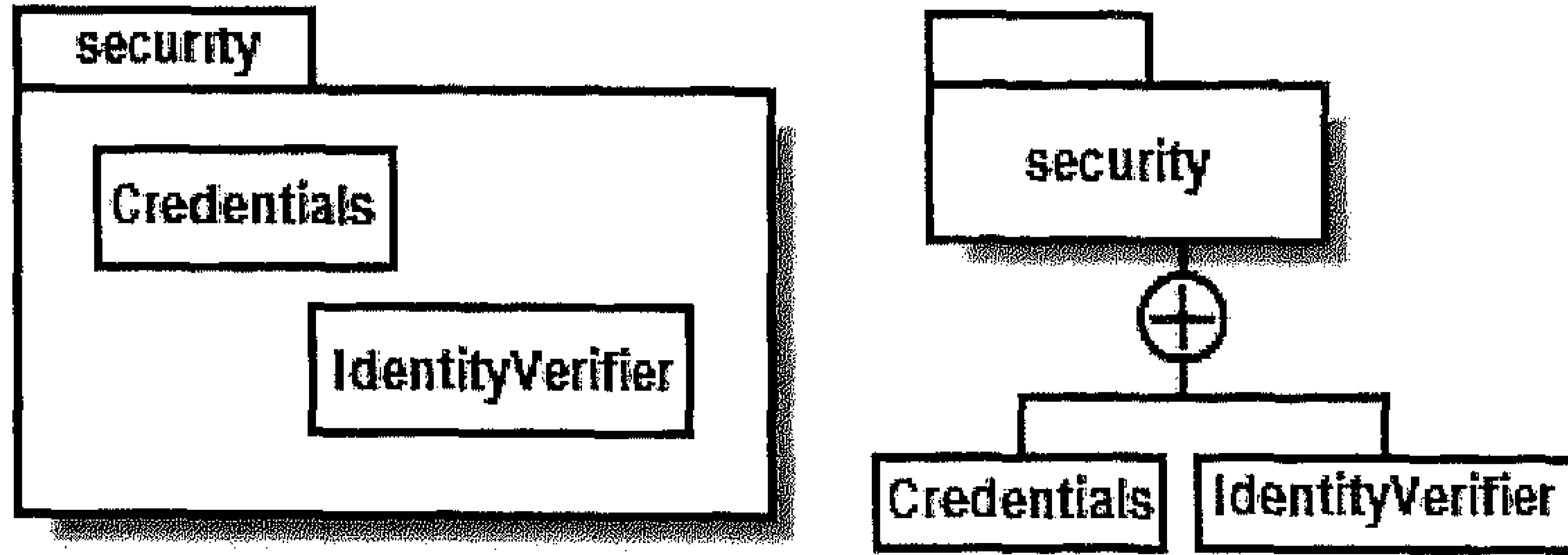


شكل رقم (١٣-٢) بعض الحُزَم في نظام إدارة محتوى CMS حيث يقابل كل حُزمة أمراً مهماً في النظام.

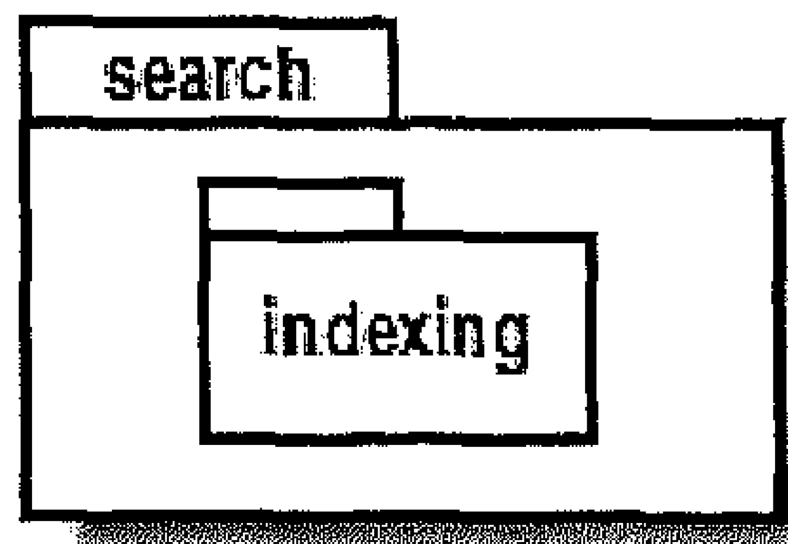
١٣-١-١ محتويات الحُزمة Contents of a Package

تقوم الحُزَم بتنظيم عناصر لغة النمذجة الموحدة (مثل الأصناف)، يمكن رسم محتوى الحُزمة بداخلها كما يمكن رسمه خارجها حيث يتم وصله بها بواسطة خط، كما هو معروض في الشكل رقم (١٣- ٣). عند رسم عناصر الحُزمة بداخلها اكتب اسم الحُزمة في عروة الحافظة.

يتم استعمال الترميز المعروض في الشكل رقم (١٣- ٣) لنمذجة أصناف لغة جافا المنتمية لحُزمة ما في جافا. تحدد الكلمة المحجوزة package بلغة جافا، التي تأتي في بداية تعريف الصنف، انتماء هذا الصنف إلى حزمة ما. يعرض المثال رقم (١٣-١) نموذج عن شفرة جافا مطابقة للصنف وثائق الاعتماد **Credentials** في الشكل رقم (١٣-٣).



شكل رقم (١٣-٣) عرض أسلوبان لبيان أن الصنفَين وثائق اعتماد **Credentials** والمتحقق من الهوية **IdentityVerifier** موجودان في الحُزمة **security**.



شكل رقم (١٣-٤) تحتوي الحُزمة **search** على الحُزمة الأخرى **indexing**.

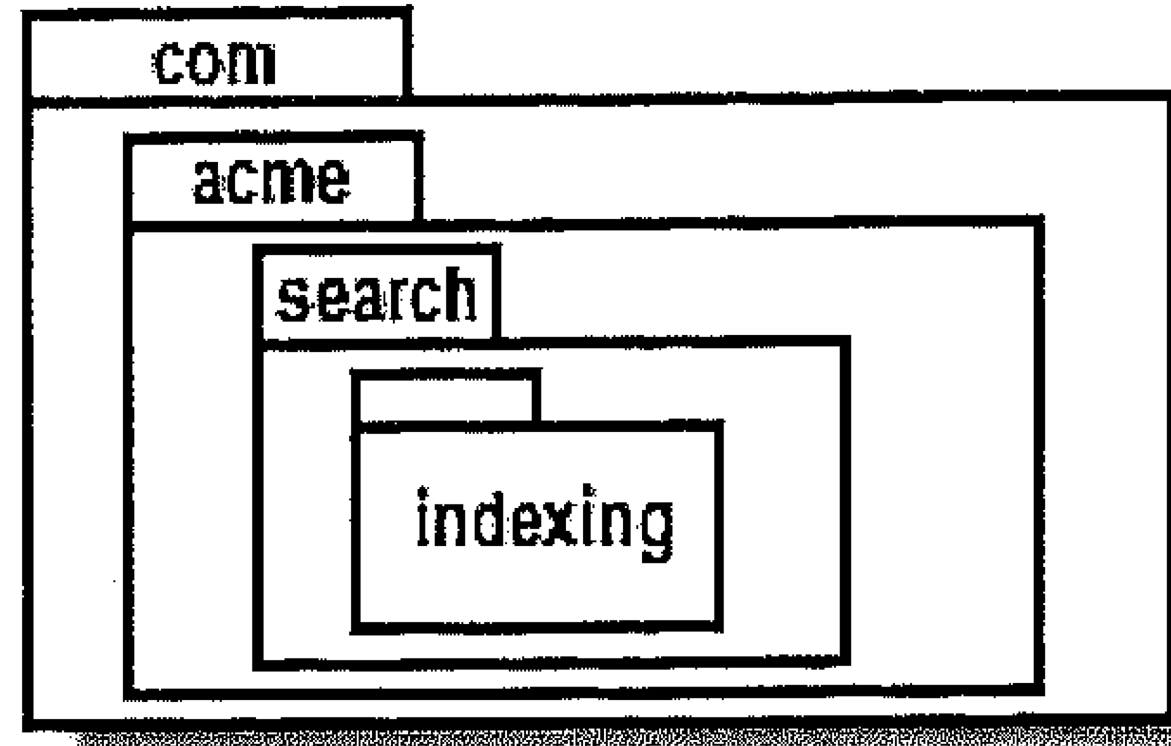
مثال رقم (١٣-١) نضيف الصنف **Credentials** إلى الحُزمة **security** من خلال شفرة جافا:

```
package security;
public class Credentials {
        ...
}
```

يمكن أن تحتوي الحُزَم أيضاً على حُزم أخرى، كما هو معروض في الشكل رقم (١٣-٤).

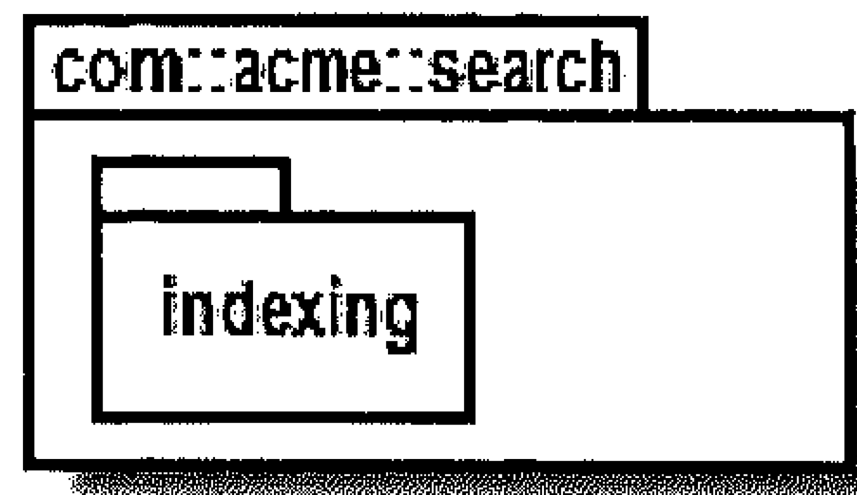
من الشائع رؤية الحُزَم متداخلة بعمق في تطبيقات المشروع. عادة ما تستعمل تطبيقات جافا عرفاً لتسمية الحُزم بالاتجاه العكسي لمحدد موقع المصدر الموحّد Uniform Resource Locator - URL (يتم أخذ أسماء الحُزم من آخر جزء في URL حتى أول جزء منه مع حذف الجزء www، حيث يستعمل رمز النقطة للفصل بين الأجزاء). على سبيل المثال، تقوم الشركة ACME التي لديها الموقع (URL) http://www.acme.com، بوضع كل

حُزمها تحت الحُزمة acme، التي تكون بدورها تحت الحُزمة com، كما هو معروض في الشكل رقم (١٣-٥).



شكل رقم (١٣-٥) من الشائع رؤية الحُزَم متداخلة بعمق في تطبيقات المشروع: عَرْضُ الحُزمتين search و indexing في هيكل نموذجي لحُزم الشركة ACME.

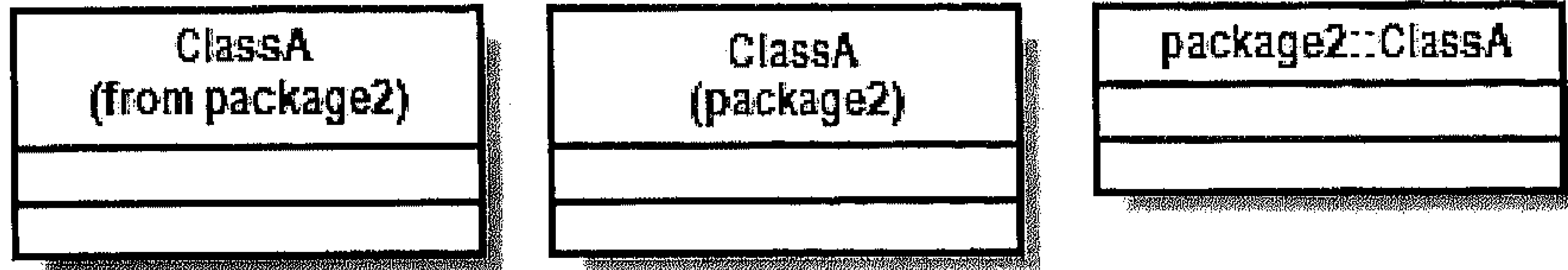
حتى هذه النقطة، تستهلك هذه الحُزَم حيزاً كبيراً، و إذا أردت عرض الأصناف داخل الحزمة فهرسة indexing، يكون على كل حزمة بداخلها التوسّع في الحجم وفقاً لذلك. ولحسن الحظ، هناك ترميز بديل أسهل في العمل هكذا حالات. ويمكن "تسطيح" الحُزَم المتداخلة وكتابتها بالصيغة packageA::packageB::packageC. ويمكن تحويل الشكل رقم (١٣-٥) إلى الشكل رقم (١٣-٦) الأقل بعثرة.



شكل رقم (١٣-٦) تسطيح الحُزَم المتداخلة.

١٣-١-٢ اختلافات أدوات لغة النمذجة الموحدة
UML Tool Variation

هناك عدد قليل من أدوات لغة النمذجة الموحدة لا تدعم حالياً الترميز المعروض في الشكل رقم (١٣-٣). وعلى أية حال، يمكن لكل الأدوات تقريباً عرض انتماء صنف ما إلى حزمة معينة باستعمال إحدى الترميزات المعروضة في الشكل رقم (١٣-٧). إن الترميز الذي عند أقصى يمين الشكل هو الترميز القياسي في لغة النمذجة الموحدة لفضاء الأسماء، وسنناقشه في القسم التالي "إشارة فضاءات الأسماء والأصناف بعضها إلى بعض".



شكل رقم (١٣-٧) الأساليب الشائعة التي تعرض فيها أدوات لغة النمذجة الموحدة انتماء صنف ما إلى حزمة معينة.

من أجل تحديد الحزمة المحتوية لصنف ما، تسمح معظم أدوات لغة النمذجة الموحدة بإدخال اسم الحزمة في صندوق حوار عن مواصفات الصنف، أو بسحب الصنف يدوياً داخل الحزمة المنتمي إليها ضمن عرض شجري البنية لعناصر النموذج.

١٣-٢ إشارة فضاءات الأسماء و الأصناف بعضها إلى بعض
Namespaces and Classes Referring to Each Other

يقدم فصل الأصناف إلى حُزَم بعض الإدارة لمواقع الحُزم. وإذا كنت مبرّمجاً بلغة جافا، ربما تكون قد صادفت سابقاً مسألة متعلقة بهذا الأمر. ولاستعمال الصنف ArrayList في برنامج جافا، عليك تحديد

أن الصنف ArrayList موجود في الحزمة java.util. هذا بسبب قيام حُزَم جافا بتعريف فضاءات الأسماء أو سياقات التسمية الخاصة بها. إذا كان العنصر غير موجود في فضاء الأسماء الحالي فعليك تحديد موقعه.

بشكل مشابه، تقوم حزمة لغة النمذجة الموحدة ببناء فضاء للأسماء. لذلك، إذا أراد عنصر في حزمة ما استعمال عنصر آخر في حزمة أخرى، يترتب عليه تحديد مكان العنصر المطلوب استعماله. ولتحديد سياق عنصر ما بلغة النمذجة الموحدة، قم بتوفير الاسم كامل المدى fully-scoped name الذي يتضمن اسم الحزمة واسم العنصر مُفرّق بينهما بنقطتَين عموديتَين مكررة (::)، كما في الصيغة packageName::className. إن الاسم كامل المدى للصنف Credentials المنتمي للحزمة security هو security::Credentials. إذا كان عندك صنفَان بنفس الاسم في حُزَم مختلفة، ويسمح لك استعمال الاسم الكامل المدى حينئذ بالتمييز بينهما.

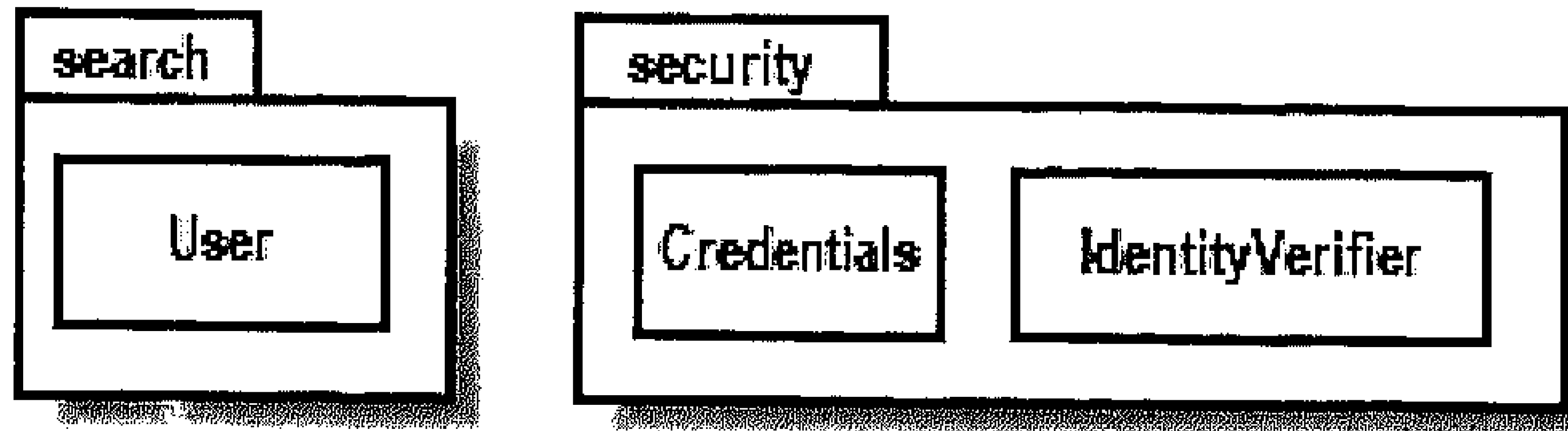
يجب أن تكون أسماء العناصر التي في نفس فضاء الأسماء فريدة بشكل منقطع النظير. هذا يعني أن الحزمة security لا يمكن أن تحتوي على صنفَين بالاسم Credentials، ولكن يمكن تواجد صنفَين بالاسم Credentials في حُزمتَين منفصلتَين، مثل الحُزمتَين security و utils. ويمكن أن تعرض أداة لغة النمذجة الموحدة الأصناف في الشكل رقم (١٣-٨) بأشكال مختلفة، كما تم مناقشته سابقاً في القسم "اختلاف أداة لغة النمذجة الموحدة".

security::Credentials　　utils:: Credentials

شكل رقم (١٣-٨) عرض الصنف باستعمال اسمه كامل المدى: تحتوي كلا الحزمتَين **security و utils على صنف بالاسم Credentials.**

لماذا الاهتمام بهذا الأمر؟ يجب تحديد فضاء أسماء ما من أجل الإشارة إلى أن صنفاً ما متعلق بصنف آخر.

تكون الأصناف التي في نفس الحزمة جزءاً من نفس فضاء الأسماء، لذلك يمكنها الإشارة بعضها إلى بعض من دون استعمال الاسم كامل المدى. وبما أنها في نفس الحزمة، يمكن أن يكون للصنف IdentityVerifier خاصية من نوع الصنف Credentials لكن ليس عليه تحديد حزمته، انظر الشكل رقم (١٣-٩).



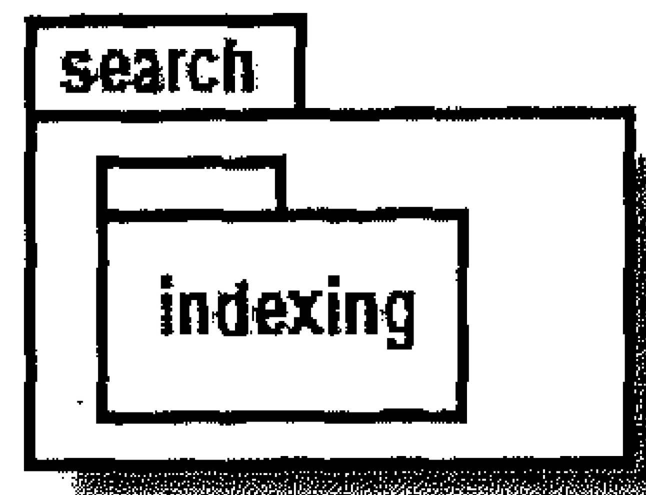
شكل رقم (١٣-٩) يجب على الأصناف الموجودة في حُزَم مختلفة توفير مدى الاسم.

من ناحية أخرى، يجب على الصنف الذي خارج الحزمة security، مثل الصنف User، توفير مدى ما عند الوصول إلى الصنف Credentials حيث يمكن إجراء ذلك باستعمال الاسم كامل المدى security::Credentials. وسنرى لاحقا في القسم "الاستيراد و الوصول إلى الحُزَم"، أن هناك أساليب أخرى لتوفير المدى بخصوص الوصول إلى صنف ما في حزمة مختلفة.



يوازي الاسم كامل المدى في جافا عملية تحديد الحزمة، أي **security.Credentials** بدلاً من مجرد **Credentials**.

في لغة النمذجة الموحدة، يمكن للعناصر التي في حزمة داخلية الإشارة إلى العناصر التي في الحزمة المحتوية لها من دون الحاجة إلى تحديد مدى الاسم، وهذا يعني أنه يمكن لعنصر ما في الحزمة indexing الإشارة إلى عنصر ما في الحزمة search من دون استعمال الاسم كامل المدى، وفقاً للشكل رقم (١٣-١٠).



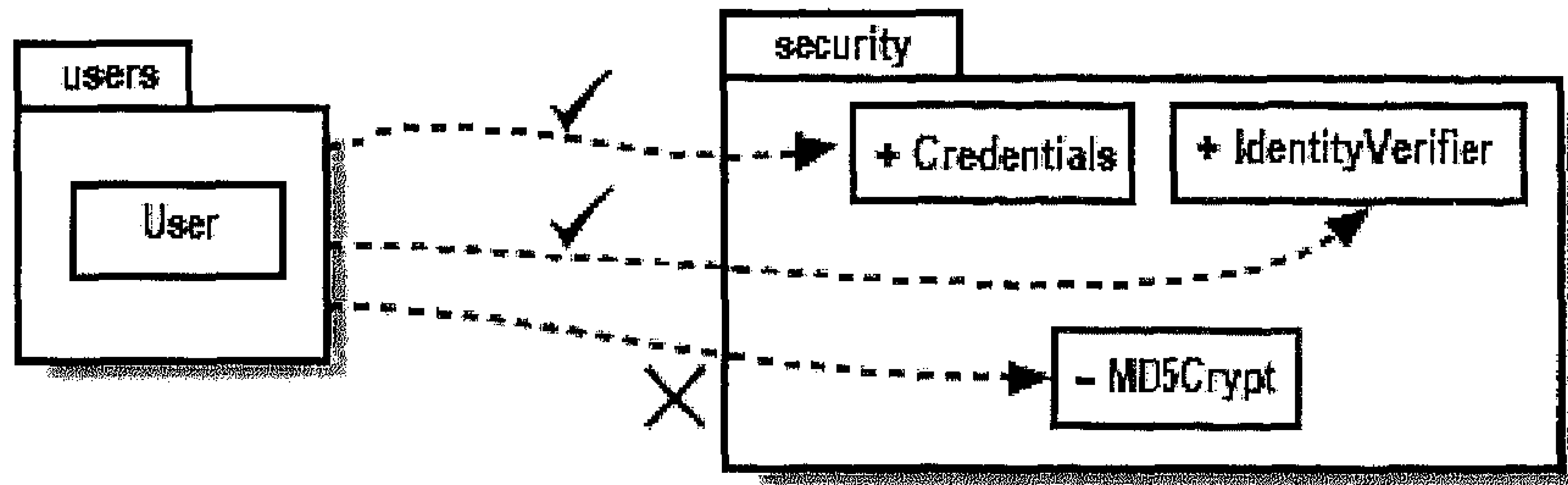
شكل رقم (١٣-١٠) في لغة النمذجة الموحدة، تدل الحزمة الداخلية على "وراثة" فضاء الأسماء الذي لا يطبق في بعض لغات البرمجة.

إن افتراض الوصول التلقائي للعناصر التي في الحزم الداخلية إلى العناصر التي في الحزم التي تحتويها لا يتوافق مع بعض لغات البرمجة. كما هو الحال في لغة جافا، إذا قام صنف ما في الحزمة indexing باستعمال صنف محدد في الحزمة search، فيجب عليه توفير المدى إما باستعمال الاسم المؤهل الكامل أو باستيراد الحزمة search. رغم حقيقة اختلاف دلالات semantics الحزم المتداخلة في لغة النمذجة الموحدة عن الحُزَم في جافا، ما زلت تستطيع استعمال الشكل رقم (١٣-١٠) لنمذجة الحزمة search.indexing في نظام جافا.

١٣-٣ رؤية العنصر Element Visibility

يمكن أن تأخذ عناصر الحزمة الرؤية العامة أو الرؤية الخاصة. إن العناصر التي لديها رؤية عامّة **public visibility** يكون الوصول إليها

متاحاً من خارج الحزمة. بينما العناصر التي لديها رؤية خاصة **private visibility** تكون متوفرة فقط للعناصر الأخرى التي داخل الحزمة. ويمكن نمذجة الرؤية العامة في لغة النمذجة الموحدة بكتابة الرمز زائد (+) أمام اسم العنصر، وكذلك الأمر بالنسبة للرؤية الخاصة حيث يستعمل هنا الرمز سالب (-)، كما هو معروض في الشكل رقم (١٣-١١).



شكل رقم (١٣-١١) بما أن العنصر **MD5Crypt** لديه رؤية خاصة، فلا يمكن الوصول إليه من خارج الحزمة **security**.

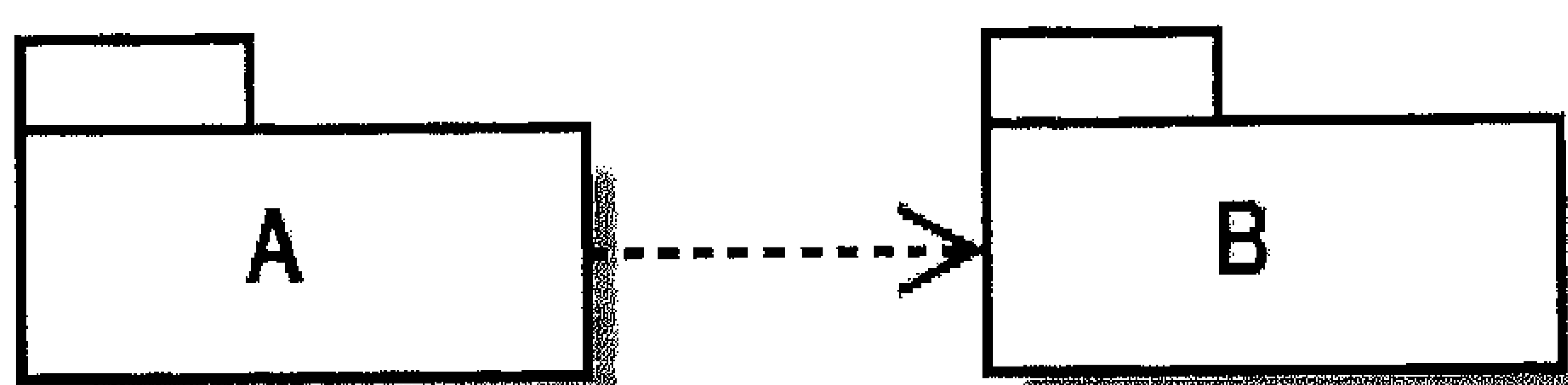
يقابل الرؤية العامة و الخاصة بلغة جافا كون الصنف عاماً أو خاصاً بالنسبة للحزمة. ويتم تحديد الصنف في جافا بأنه عام بالنسبة للحزمة من خلال محدد الوصول public، كما في الشفرة التالية:

```
public class Credentials { }
```

إذا كانت الكلمة الأساسية public غير مذكورة، فيكون الصنف خاصاً بالنسبة للحزمة. إن العديد من أدوات لغة النمذجة الموحدة لا توفر رمز الزائد ورمز السالب لعرض رؤية العناصر، لذلك لا تتفاجأ إذا لم تكن متوفرة لديك.

١٣-٤ اعتمادية الحزمة Package Dependency

لقد أظهرت الأقسام السابقة حاجة صنف ما في حزمة محددة إلى استعمال صنف في حزمة أخرى. ويسبب ذلك علاقة اعتمادية بين الحُزَم: إذا كان عنصر في الحزمة A يستعمل عنصرا في الحزمة B، نقول حينئذ أن الحزمة A تعتمد على الحزمة B، كما هو معروض في الشكل رقم (١٣-١٢).



شكل رقم (١٣-١٢) تعتمد الحزمة A على الحزمة B.

الحُزَم في البرامج Packages in Your Software

بعد دراسة أساسيات مخططات الحزم، حان الوقت للتفكير بسبب الرغبة باستعمال الحُزَم في البرامج.

إذا كنت تقوم بإنشاء برنامج صغير جداً (يتألف من بضعة أصناف فقط)، قد لا تهتم بتنظيم أصنافك في حُزَم. لكن عندما يكبر برنامجك و تقوم بإضافة المطورين إلى المشروع، تقدم الحُزَم هيكلة التنظيم وتسمح بمعرفة مَنْ يعمل على ماذا.

يمكن أن تكون الشفرة المتعلقة بواجهة المستخدم الرسومية (GUI) موجودة في الحزمة gui، وتكون الشفرة المتعلقة بقدرات البحث موجودة في الحزمة search، وتكون الخدمات العامة موجودة في الحزمة util. ويسهّل هذا الأمر التحري عن الأصناف عند البحث في واجهة برامج تطبيقية معقدة (API). على سبيل المثال، كي تحدد مكان نافذة

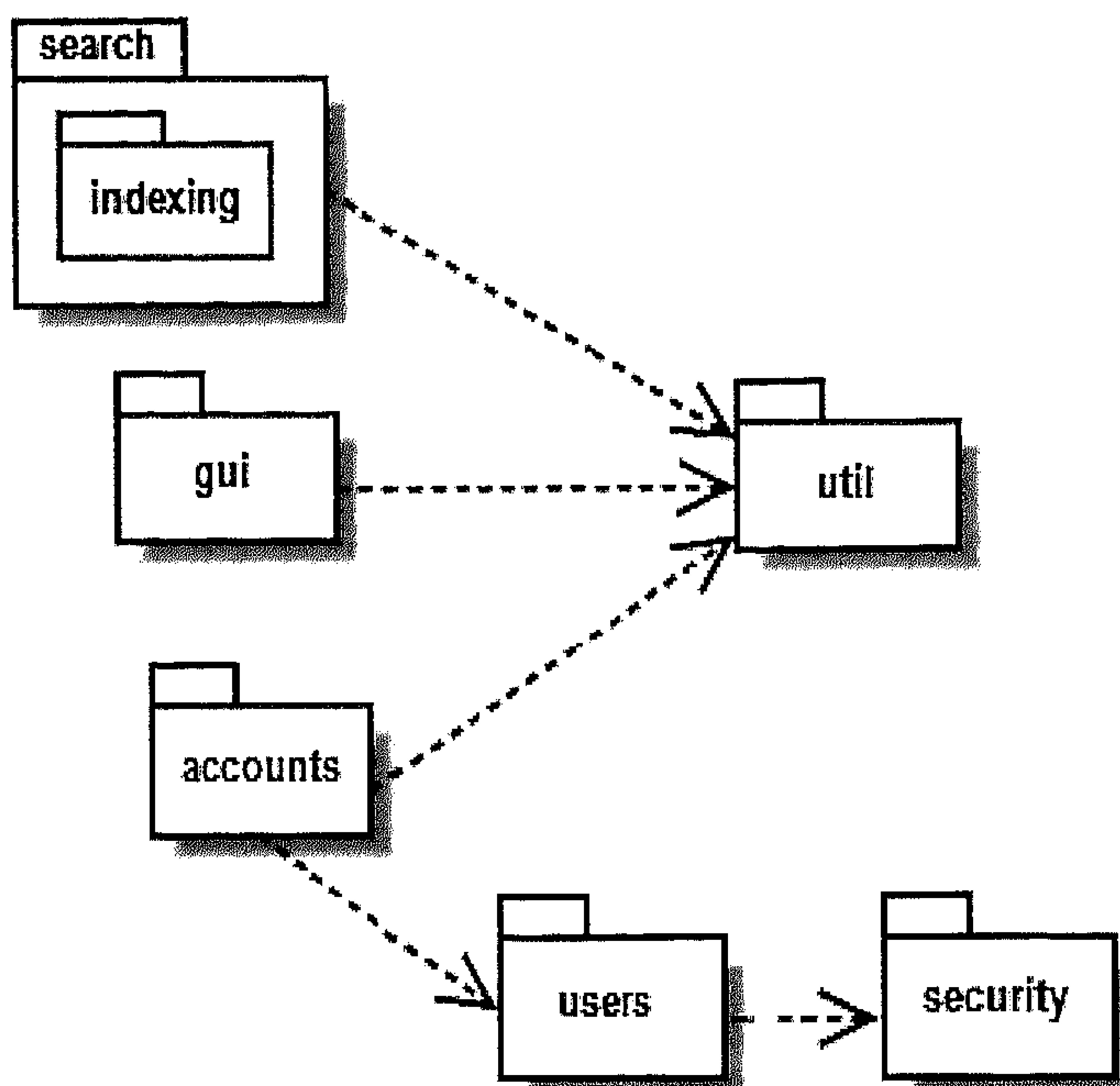
حوار ما في عناصر الواجهات الرسومية GUI، عليك معرفة البحث في الحزمة gui.

وغالباً ما يعمل المبرمجون على حُزمهم أو حُزم فريقهم بكل سجية دون انزعاج. ولا يقوم -عادة- العاملون على الحزمة gui بتغيير في الحزمة search والعكس بالعكس. ويمكن لأي شخص استعمال الحُزَم شائعة الاستخدام (مثل الحزمة util)، لكن من المتوقع أن تكون تلك الحُزَم مستقرة تماماً لأنه قد تؤثر التغييرات على كل منها. بالإضافة إلى تنظيم العناصر، ويمكن أن تقدم الحُزَم وظائف أخرى مفيدة؛ يمكن استعمالها للتحكم بالوصول: يمكن التصريح عن عناصر بأنها خاصة بالنسبة لحزمة ما لحمايتها و منع الوصول إليها و استعمالها من قبل الحُزَم الأخرى.

ويمكن أن تساعد الحُزَم على تنظيم الأصناف على شكل وحدات برمجية للنشر. على سبيل المثال، إذا أردنا تضمين قدرات البحث في بعض الأنظمة دون سواهم، فيمكن اختيار تضمين أو استبعاد حزمة البحث search في بنية الأنظمة.

يفيد فهم الاعتمادية بين الحُزَم لأجل تحليل استقرار البرامج، كما تمت مناقشته في القسم "إدارة اعتماديات الحُزَم". في الحقيقة، أن معظم الاستعمال الشائع لمخططات الحُزَم بلغة النمذجة الموحدة ، لتوفير مُلخص عن الحُزَم الرئيسية في البرامج و الاعتماديات التي بينها ، كما هو معروض في الشكل رقم (١٣-١٣).

يعرض القسم "إدارة اعتماديات الحُزَم" ، الذي يأتي لاحقاً في هذا الفصل، مرة ثانية مخططات اعتماديات الحُزَم، وذلك لتبيان كيفية استعمالها لفهم وتحسين استقرار البرامج.



شكل رقم (١٣-١٣) مخطط حُزم نموذجي يعرض الحُزَم الأساسية واعتماداتها.

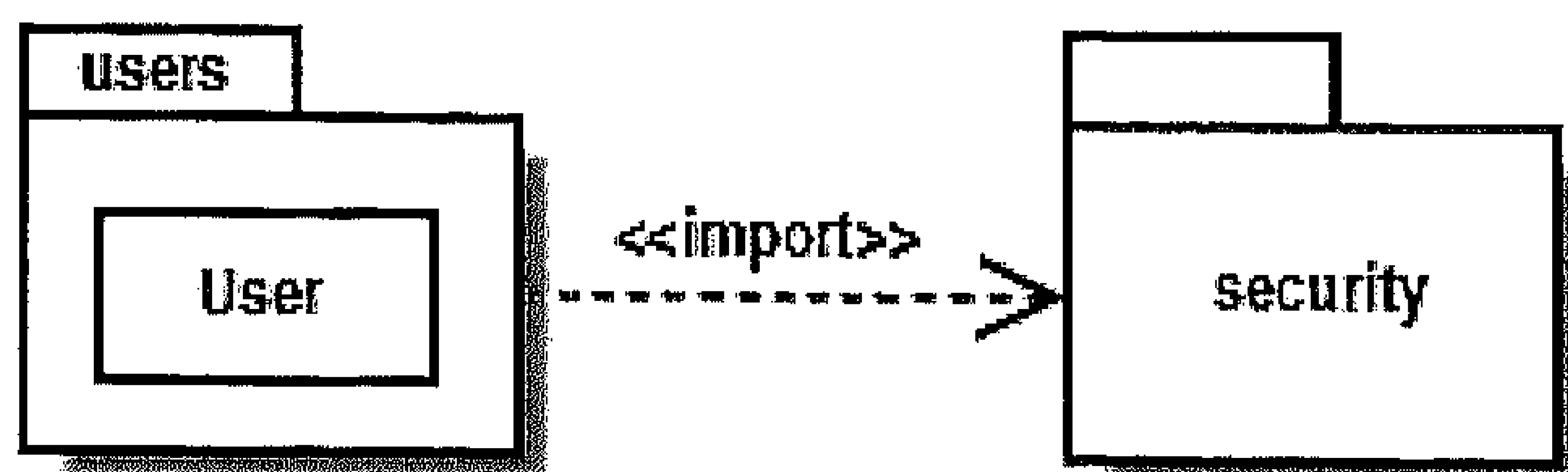
١٣-٥ استيراد الحُزَم و الوصول إليها
Importing and Accessing Packages

عندما تقوم حزمة ما باستيراد **imports** حزمة أخرى، يمكن لعناصر الحزمة التي قامت بالاستيراد استعمال عناصر الحزمة التي تم استيرادها، وذلك من دون الحاجة إلى استعمال أسمائهم كاملة المدى. وتشبه هذه الميزة عملية الاستيراد مع التعليمة import بلغة جافا، حيث يمكن لصنف ما استيراد حزمة و استعمال محتوياتها من دون الحاجة إلى تزويد أسماء حُزمهم.

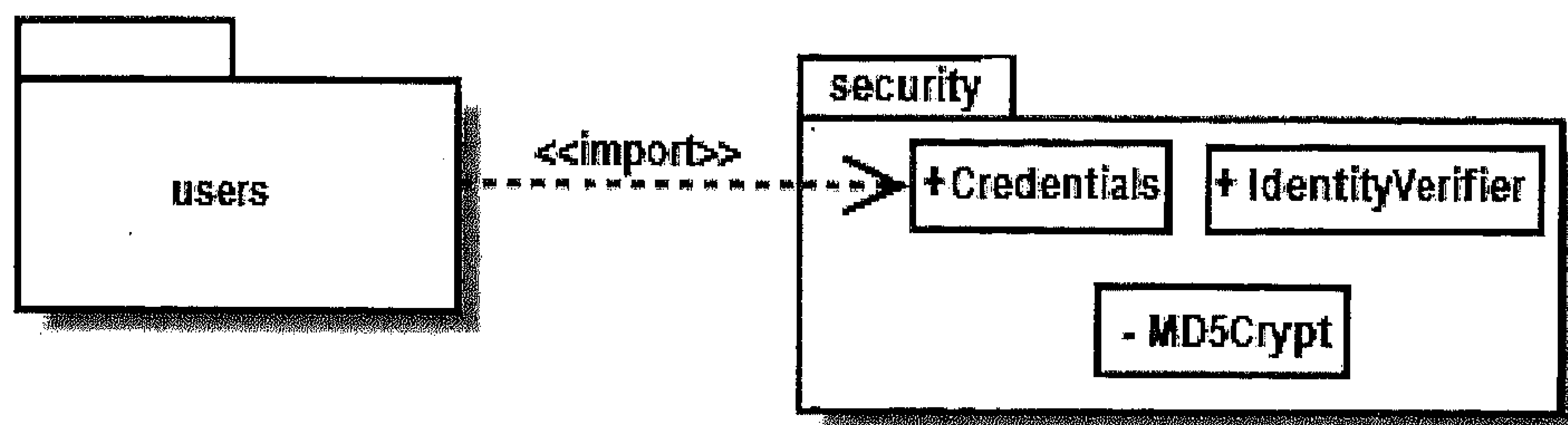
في علاقة الاستيراد، يشار إلى الحزمة التي تم استيرادها بالحزمة الهدف **target package**. ولإظهار علاقة الاستيراد نقوم برسم سهم

اعتمادية من الحُزمة التي قامت بالاستيراد إلى الحزمة الهدف مع استعمال الحاشية <<import>>، كما هو معروض في الشكل رقم (١٣-١٤).

يمكن أيضاً لحزمة ما استيراد عنصر محدد في حزمة أخرى بدلاً من استيراد الحزمة كاملة، كما هو معروض في الشكل رقم (١٣-١٥).

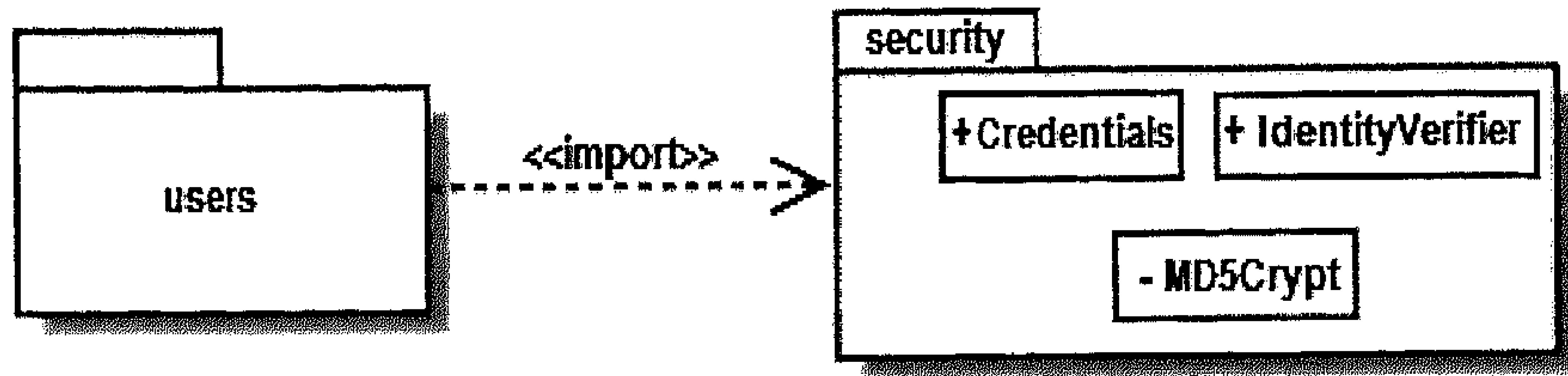


شكل رقم (١٣-١٤) تقوم الحزمة **users** باستيراد الحزمة **security**، لذلك يمكن لأصناف الحزمة **users** استعمال الأصناف العامة في الحزمة **security** من دون الحاجة إلى تحديد اسمها.



شكل رقم (١٣-١٥) تستورد الحزمة **users** العنصر **Credentials** فقط من الحزمة **security**.

عند استيراد حزمة محددة، تكون العناصر العامة في الحزمة الهدف هي المتوفرة فقط في فضاء الأسماء الذي قام بالاستيراد. ولنأخذ الشكل رقم (١٣-١٦) على سبيل المثال، يمكن لعناصر الحزمة users رؤية العنصرَين Credentials و IdentifyVerifier ولكن لا يمكنها رؤية العنصر MD5Crypt.



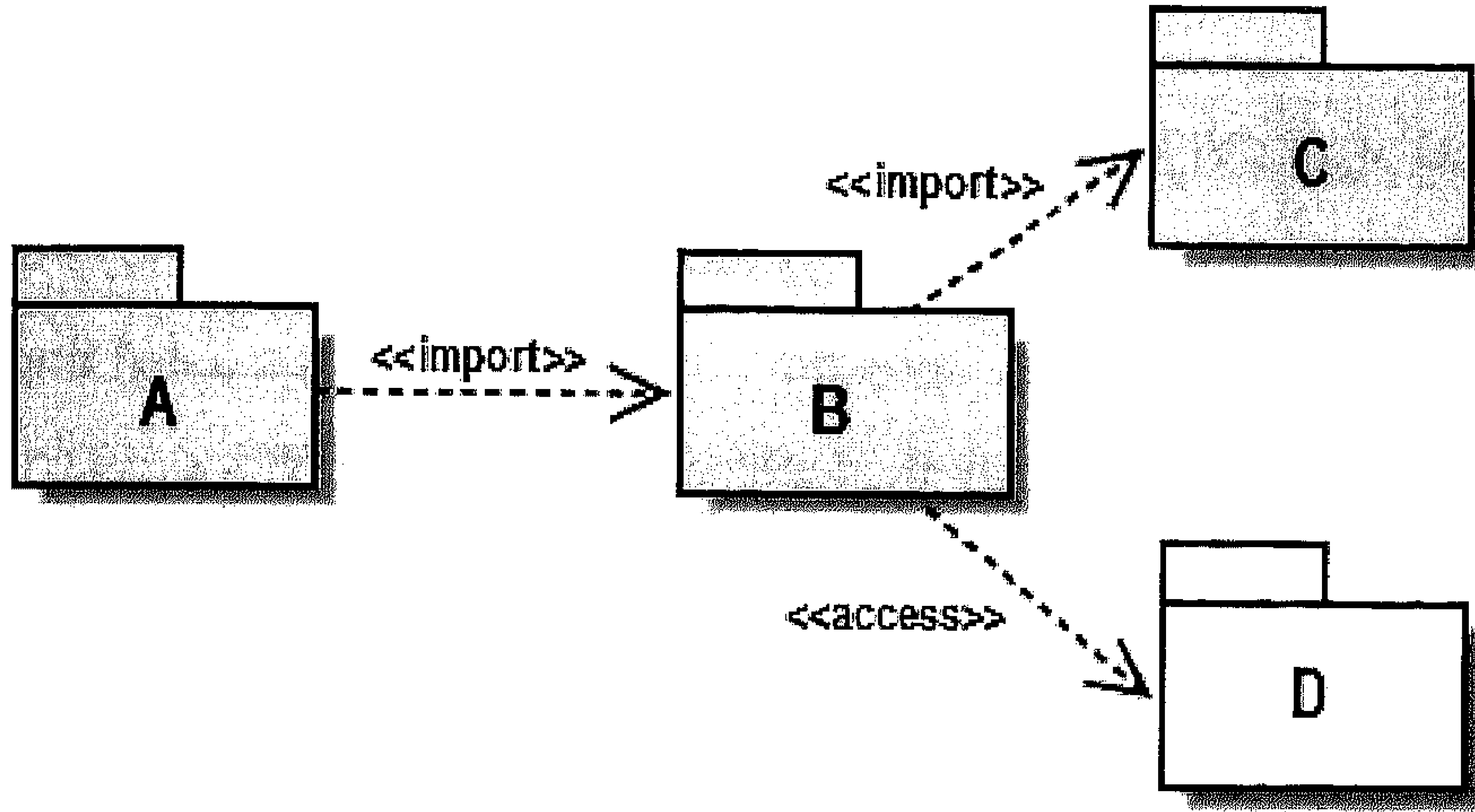
شكل رقم (١٣-١٦) تتسبب الرؤية الخاصة للصنف بعدم رؤيته حتى إن كانت حزمته مستوردة.

لا تقتصر الرؤية على العناصر فقط، فلعلاقة الاستيراد نفسها أيضاً رؤية خاصة بها. ويمكن أن يكون الاستيراد استيراداً عاماً public import أو استيراداً خاصاً private import، حيث يكون الاستيراد الافتراضي هو الاستيراد العام. ويعني الاستيراد العام أن العناصر المستوردة لها الرؤية عامة داخل فضاء الأسماء الذي قام بالاستيراد، ويعني الاستيراد الخاص أن العناصر المستوردة لها الرؤية خاصة في فضاء الأسماء الذي قام بالاستيراد. ويتم عرض الاستيراد الخاص باستعمال الحاشية <<access>> بدلاً من الحاشية <<import>>.

ويظهر الاختلاف بين الاستيراد **import** و الوصول إلى **access** عند قيام حزمة ما باستيراد حزمة معيّنة، حيث تقوم هذه الأخيرة بدورها بالوصول إلى حُزم أخرى. وتأخذ العناصر المستوردة الرؤية عامة في الحزمة التي قامت باستيرادها، و هكذا تُصبح الرؤية تنتقل مع الاستيرادات الإضافية، بينما لا يتم ذلك مع العناصر التي يتم الوصول إليها.

في الشكل رقم (١٣-١٧)، تقوم الحزمة B باستيراد الحزمة C والوصول إلى الحزمة D، لذلك يمكن أن ترى الحزمة B العناصر العامة التي في الحزمة C والحزمة D. من ناحية أخرى، تقوم الحزمة A باستيراد الحزمة B، لذلك يمكن أن ترى الحزمة A العناصر العامة التي في الحزمة

B. كما يمكن أن ترى الحزمة A أيضاً العناصر العامة التي في الحزمة C لأن الحزمة C قد تم استيرادها مع الرؤية عامة من قِبَل الحزمة B، ولكن لا تستطيع الحزمة A رؤية أي شيء في الحزمة D لأن الحزمة D قد تم استيرادها (الوصول إليها) مع الرؤية خاصة من قِبَل الحزمة B.



شكل رقم (١٣-١٧) يمكن أن ترى الحزمة A العناصر العامة في الحزمة C ولا ينطبق ذلك على الحزمة D.

يمكن استعمال علاقات الاستيراد و الوصول لنمذجة مفاهيم استيراد الأصناف إلى فضاء أسماء آخر في عالم البرمجة، وذلك كي تستطيع عناصر فضاء الأسماء الذي قام بالاستيراد الإشارة إلى عناصر فضاء الاسم الهدف من دون استعمال الاسم كامل المدى. على سبيل المثال، يمكن استعمال علاقات الحُزَم في الشكل رقم (١٣-١٤) لنمذجة شفرة جافا في المثال رقم (١٣-٢).

يوازي عنصر الاستيراد import الذي في الشكل (١٣-١٥) شفرة جافا المعروضة في المثال رقم (١٣-٣).

لا يهتم عديد من المنمذجين بتحديد علاقات الاستيراد و الوصول، حيث إنهم يظهرون اعتماديات الحُزم المعمّمة generic المناقشة مؤخراً في القسم "اعتماديات الحُزَم".

مثال رقم (١٣-٢) بما أن الصنف User يستورد كامل الحزمة security، بإمكانه إذن الإشارة إلى الصنف Credentials من دون استعمال الاسم المؤهل الكامل security.

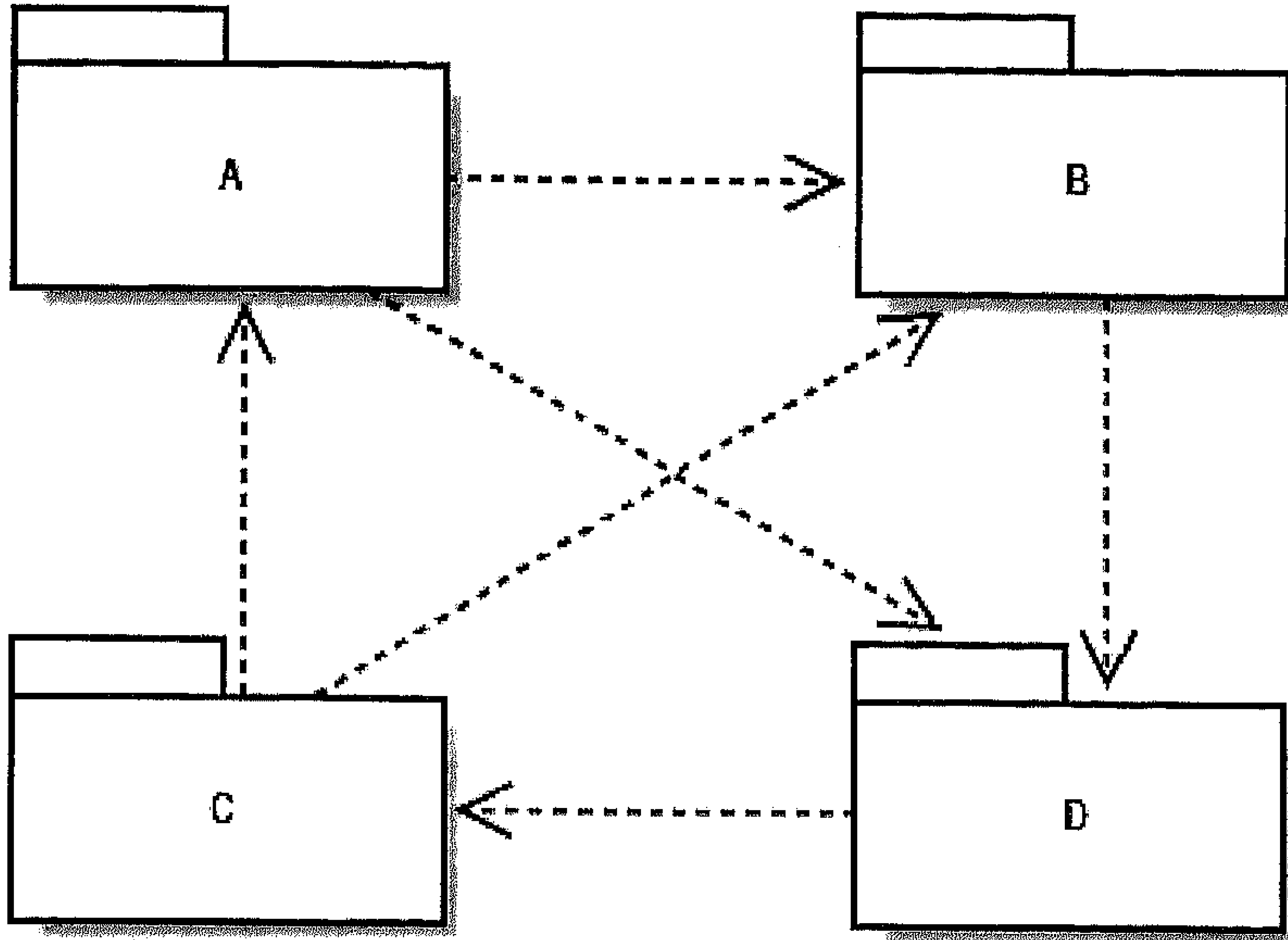
```
package users;
 // security استورد كل العناصر العامة في الحزمة //
import security.*;
class User {
    Credentials credentials;
     . . .
}
```

مثال رقم (١٣-٣) يتم هنا استيراد الصنف Credentials فقط من الحزمة security.

```
package users;
 // Credentilas استورد فقط الصنف //
import security.Credentials;
class User {
    Credentials credentials;
     . . .
}
```

١٣-٦ إدارة اعتماديات الحُزم
Managing Package Dependencies

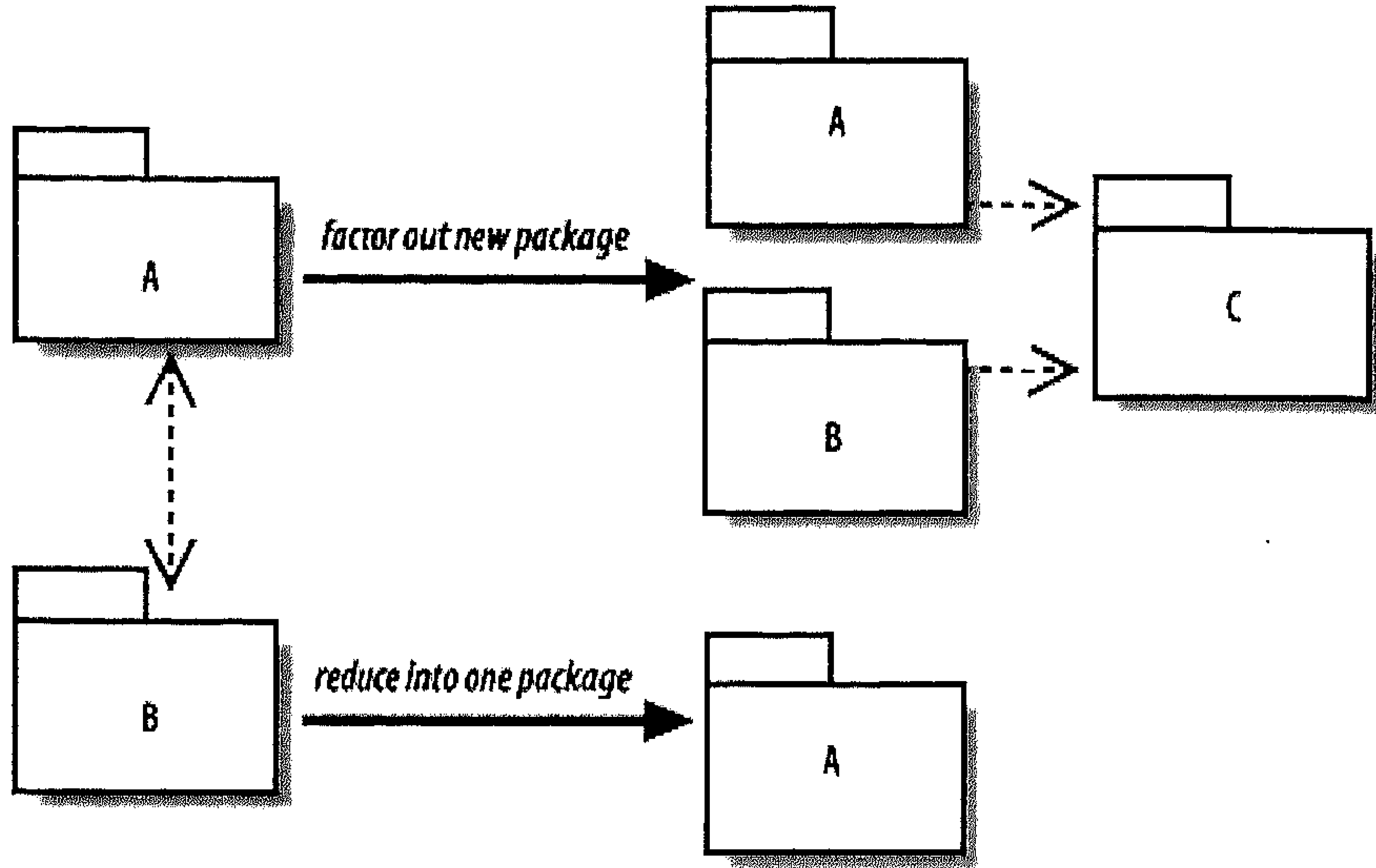
يمكن أن تؤدي الاعتماديات المعقدة بين الحُزَم إلى برامج هشّة، لأنه قد يسبب التغيير في حزمة ما إلى فشل الحُزَم المعتمدة عليها. ويعرض الشكل رقم (١٣-١٨) اعتمادية كارثية: قد يؤثر التغيير في أي حزمة على الحزم الأخرى بشكل أساسي.



شكل رقم (١٣-١٨) قد يؤثر التغيير في أي حزمة على الحزم الأخرى بشكل مباشر أو غير مباشر.

أسس روبرت مارتن، في كتابه Agile Software Development (Prentice Hall)، مبادئ و نظريات تتعلق بالاعتماديات بين الحُزَم وبوحدات النشر. مثل اجتناب الاعتماديات الدورية مع الحزم و بالاعتماد على "اتجاه الاستقرار"، حيث يمكن اكتشافهما بالنظر في مخططات الحُزَم.

إذا كانت الاعتماديات دورية، فيمكن كسر دوريتها بأساليب مختلفة. وإذا كان هناك اعتمادية دورية بين الحزمتَين A و B، فيمكن إنشاء حزمة جديدة مستمدة منهما بحيث تكون كلا الحزمتَن A و B تعتمد عليها، كما يمكن اعتبار أن كل الأصناف تنتمي لبعضها بأية حال (أي تنتمي لنفس الحزمة)، كما هو معروض في الشكل رقم (١٣-١٩).



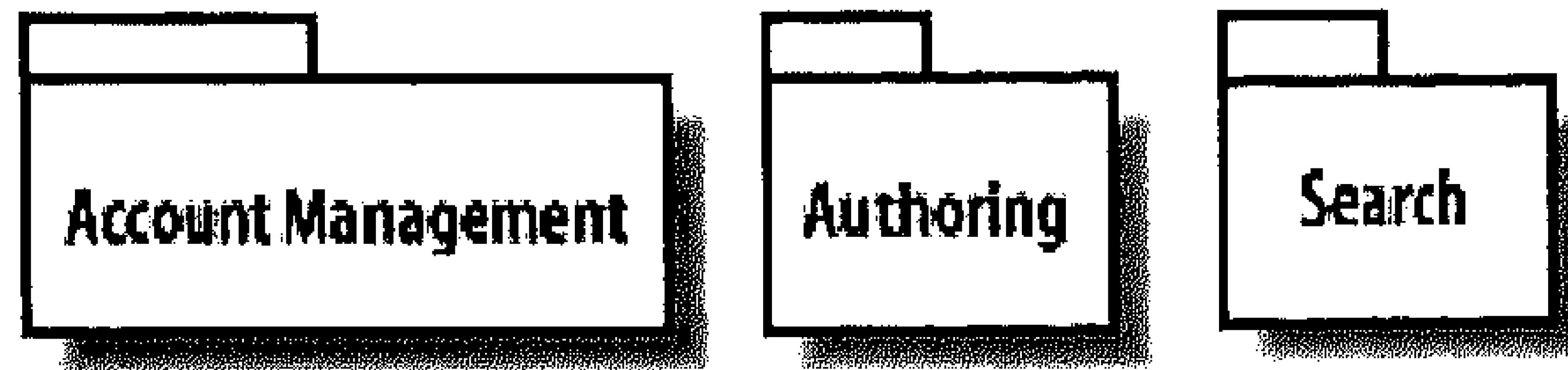
شكل رقم (١٣-١٩) إزالة دورية اعتماديات الحُزم.

يعني الاعتماد على درجة الاستقرار أنه يجب اعتماد حزمة ما على حُزَم أكثر استقراراً منها فقط. وتعتمد الحزمة غير المستقرة على كثير من الحُزَم الأخرى؛ و تعتمد الحزمة المستقرة على قليل من الحُزَم. ويمكن أن تساعد دراسة مخططات الحزمة على اكتشاف التصاميم الضعيفة المحتملة الناتجة عن الحُزَم الرئيسية في النظام (مثل الحُزَم المحتوية على واجهات) بالاعتماد على الحُزَم غير المستقرة.

١٣-٧ استعمال الحُزَم لتنظيم حالات الاستخدام

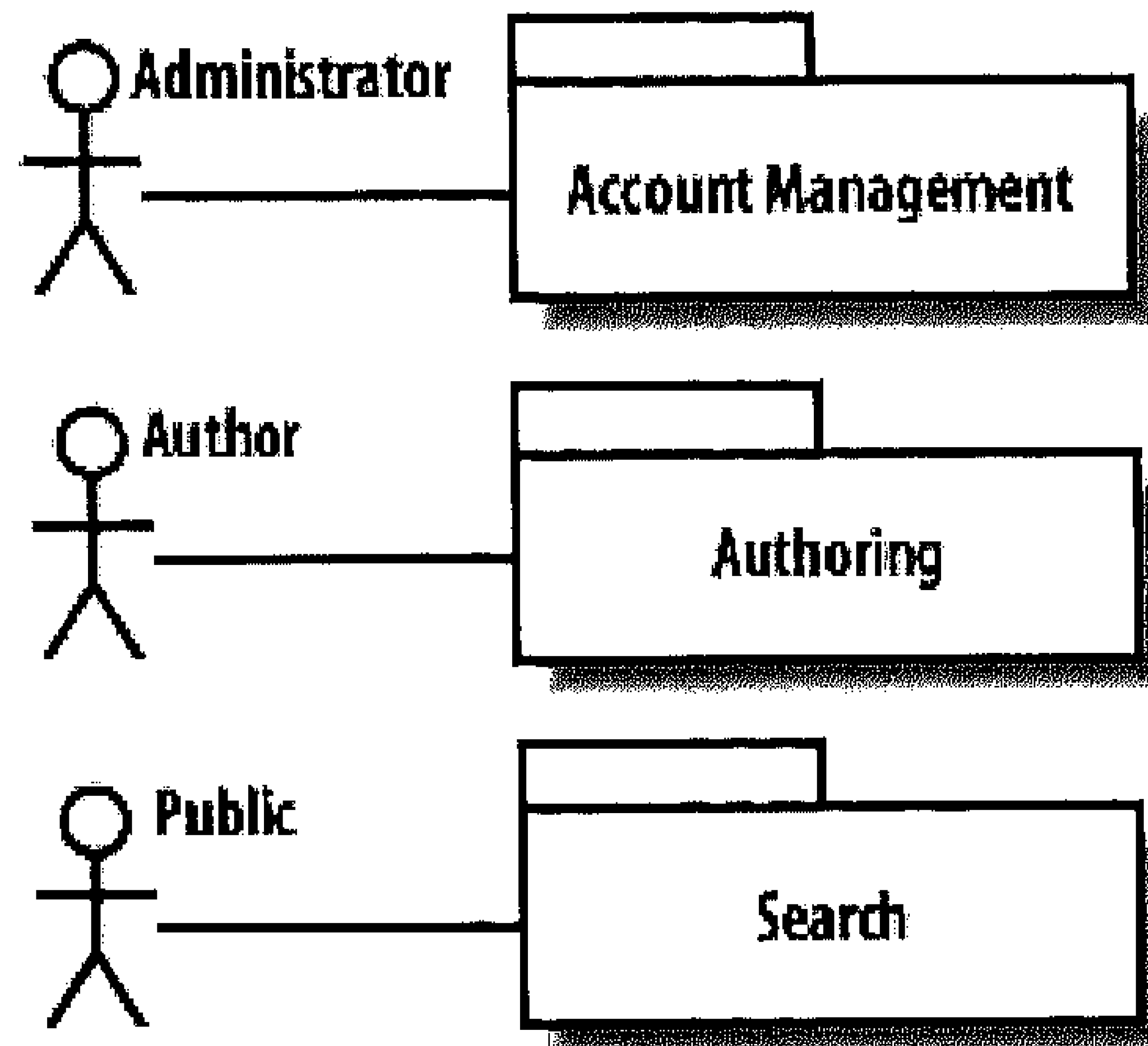
Using Packages to Organize Use Cases

بما أن الحُزَم تقوم بتجميع الأصناف المتشابهة و ظيفياً، فهي تقوم أيضا بتجميع عناصر أخرى من لغة النمذجة الموحدة كحالات الاستخدام. ويعرض الشكل رقم (١٣-٢٠) بعض حُزَم حالات الاستخدام من نظام إدارة المحتوى.



شكل رقم (١٣-٢٠) تحزيم مجموعات حالات استخدام رئيسية في نظام إدارة المحتوى.

يمكن أن تساعد ترقية حالات الاستخدام إلى مستويات أعلى في النظام على تنظيم النموذج، وذلك بالسماح برؤية أي مستخدمين يتفاعلون مع أي أجزاء في النظام، كما هو معروض في الشكل رقم (١٣-٢١).



شكل رقم (١٣-٢١) تُمكّن الحُزَم من رؤية كيفية تفاعل المستخدمين مع النظام بمستوى أعلى.

١٣-٨ ما هي الخطوة التالية؟

يتم استعمال الحُزَم لتجميع عناصر لغة النمذجة الموحدة، مثل الأصناف و حالات الاستخدام. ربما تريد مراجعة تلك الفصول للحصول على تفاصيل أكثر حول عرض محتويات الحُزمة. لقد تم تغطية مخططات

الأصناف في الفصل الرابع؛ و تم تغطية مخططات حالة الاستخدام في الفصل الثاني.

وأحد أهم تطبيقات مخططات الحُزَم هي رؤية الاعتماديات التي في النظام. تشمل المخططات عالية المستوى الأخرى المهمة للنظام، ومخططات المكوّنات التي تعرض الأجزاء الرئيسية للبرامج، ومخططات النشر التي تعرض كيفية نشر الأجزاء على الأجهزة. لقد تم وصف مخططات المكوّنات في الفصل الثاني عشر؛ وستتم تغطية مخططات النشر في الفصل الخامس عشر.

نمذجة حالة الكائن: مخططات حالة الآلة

MODELING AN OBJECT'S STATE: STATE MACHINE DIAGRAMS

تفيد مخططات النشاط ومخططات التفاعل في وصف السلوكيات، لكن ما زالت هناك أجزاء مفقودة. وأحياناً ما تكون حالة الكائن أو النظام عاملاً مهماً في سلوكياته. على سبيل المثال، إذا كان نظام إدارة المحتوى يتطلب مستخدمين قادرين على تقديم طلب حساب، الذي قد تتم المصادقة عليه أو رفضه، ثم قد يتصرف الكائن AccountApplication بشكل مختلف بالاعتماد على إذا ما كان مُعلّقاً أو مقبولاً أو مرفوضاً.

في مثل هذه الحالات، من المفيد نمذجة حالات الكائن و الأحداث المتسببة في تغيير حالته(هذا ما تعمله مخططات حالة الآلة على أفضل وجه). لنتابع المثال السابق، ربما يكون للكائن **AccountApplication** الحالات معلّق **pending**، مقبول **accepted**، ومرفوض **rejected** كقيم محتملة لإحدى خاصياته، وقد تتغير حالاته عند حصول الأحداث، مثل وافق **approve**، أو رفض **reject**. ويسمح مخطط حالة الآلة بنمذجة تلك السلوكيات.

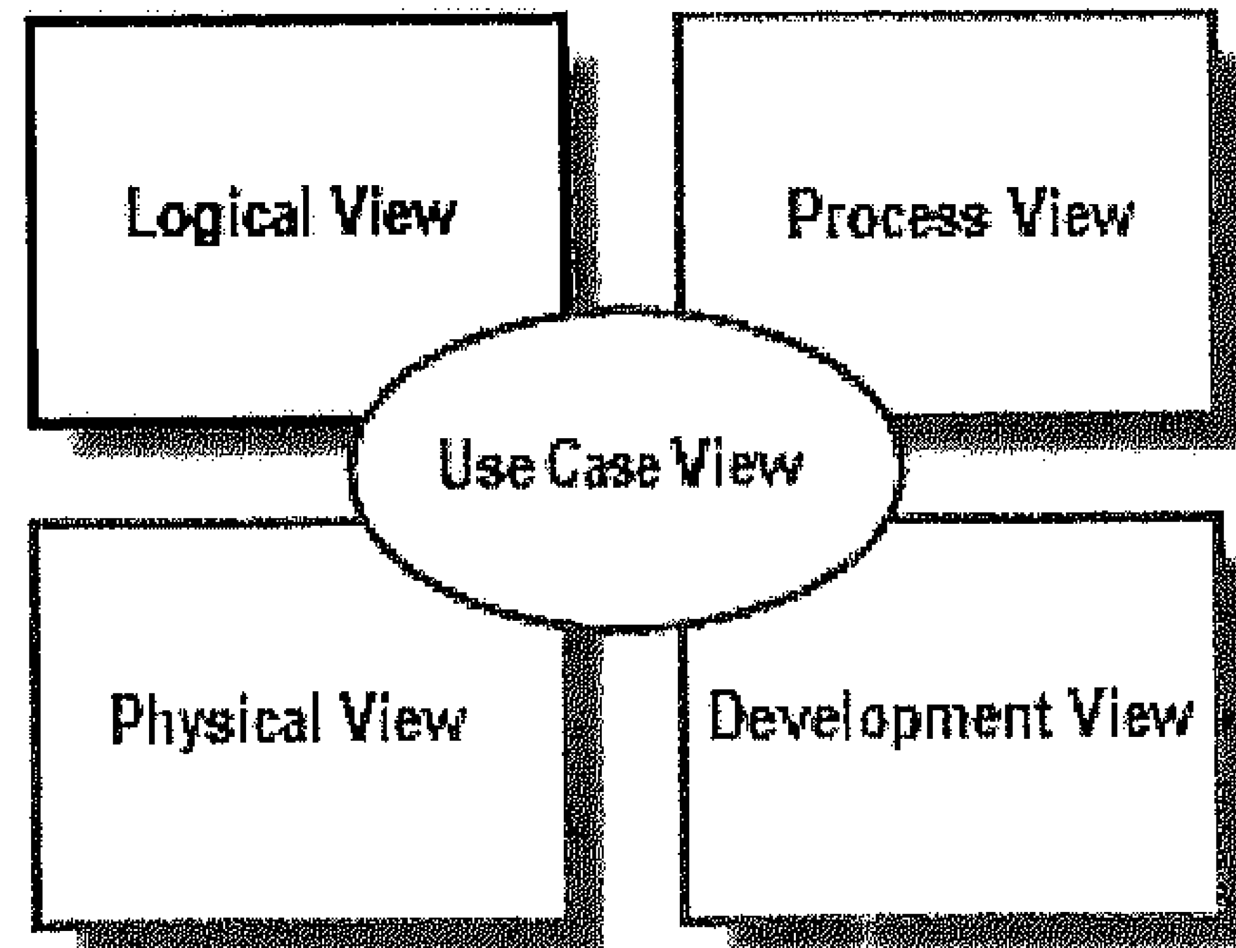
وتستعمل مخططات حالة الآلة بكثرة في مواضع خاصة بأنظمة الأجهزة والبرمجيات، بما فيها التالي:

- الأنظمة الفورية أو الأنظمة حرجة المهمة، مثل برامج مراقبة القلب.
- أجهزة خاصة تكون سلوكياتها مُعرّفة وفقاً للحالة، مثل الصراف الآلي ATM.
- ألعاب الرماية بمنظور الشخص الأول First-Person Shooter، مثل لعبة الموت دووم **Doom** أو لعبة نصف الحياة **Half-Life**.

للتفكير ملياً بهذه الاستعمالات الشائعة، سينحرف هذا الفصل عن مثال نظام إدارة المحتوى المستعمل في مُجمل بقية هذا الكتاب.

يركز معظم هذا الفصل على سلوكية آلات الحالة، التي يمكن أن تعرض الحالات والانتقالات، و السلوك (داخل الحالات و قرب الانتقالات). وهناك نوع آخر من آلة الحالة تسمى بروتوكول آلة الحالة الذي لا ينمذج السلوك إلا أنه مفيد في نمذجة البروتوكولات، مثل بروتوكولات شبكة الاتصالات. سيناقش بروتوكول آلات الحالة بشكل موجز في نهاية الفصل.

وتشكل مخططات آلة الحالة جزءاً من النموذج المنطقي للنظام، كما هو معروض في الشكل رقم (١٤-١).

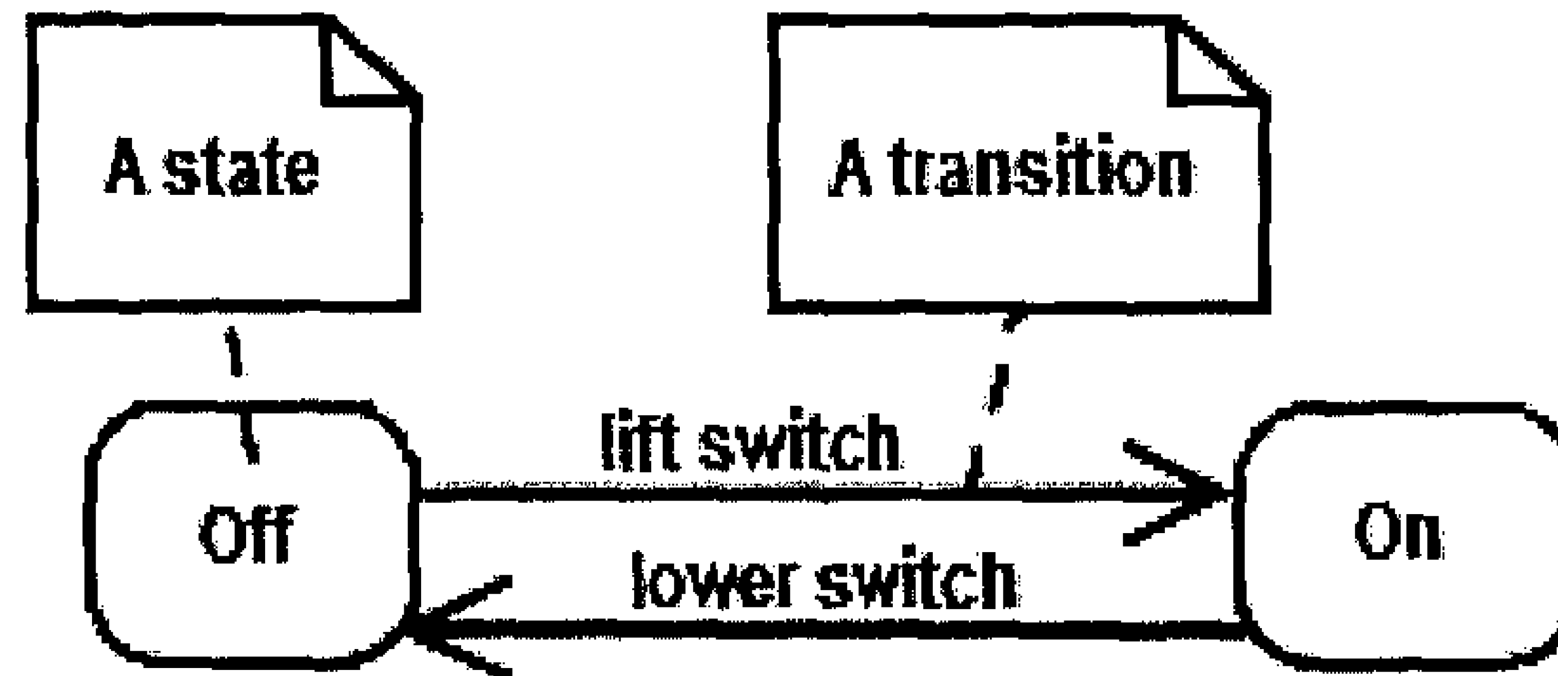


شكل رقم (١٤-١) يصف المنظور المنطقي التوصيفات المجردة لأجزاء النظام حيث تشمل متى وكيف تكون تلك الأجزاء بحالات مختلفة، باستعمال مخططات آلة الحالة.

عـادة مـا يشـار إلى مخططـات آلـة الحالـة بشـكل غـير رسمـي كمخططـات الحالـة. وربمـا تكـون قـد رأيـت أنـه يشـار إليهـا في الماضـي كمخططات خارطة الحالة، بسبب خضوع هـذا المخطـط لتغيير كثير في اسمه.

١٤-١ الأساسيات Essentials

دعنـا ننظـر إلى العناصـر الرئيسـة في مخططـات الحالـة باسـتعمال مثـال بسـيط. يعـرض الشـكل رقـم (١٤-٢) مخطـط حالـة لنمذجـة الضـوء. عنـدما ترفـع مفتـاح الضـوء light switch فيصـبح الضـوء مضـاءً on. وعنـدما تُنزل مفتاح الضـوء lower switch فيصبح الضوء مطفأً off.



شكل رقم (١٤-٢) العناصر الأساسية لمخطط الحالة: الحالات والانتقالات بينها.

يتـألف مخطـط الحالـة مـن حـالات و انتقـالات، ترسـم الحـالات كمستطيلات مـدوّرة الزوايا و ترسـم الانتقـالات كأسـهم تـربط الحـالات بعضها ببعض. ويمثل الانتقال تغيّراً في الحالة، أو كيفية الانتقال مـن حالـة ما إلى الحالـة الـتي تليها. وتصبح الحالـة نشـطة **active** عنـدما يتم الـدخول فيها من خلال انتقال ما، وتصبح الحالة خاملة **inactive** عندما يتم الخروج منها من خلال انتقال ما.

وتتم كتابة الحدث المتسبب في تغيير الحالة (أو المُطلق trigger) بمحاذاة سهم الانتقال. ويشتمل الضوء في الشكل رقم (١٤-٢) على حالتَين: الحالة مطفأ off والحالة مضاء on. وتتغيّر حالته عند إثارة المُطلقَين رفع المفتاح **light switch** أو إنزال المفتاح **lower switch**.

إذا لم تشاهد مخططات الحالة سابقاً، فمن المفيد مشاهدة الحالات و الانتقالات على شكل جدول، كما هو معروض في الجدول رقم (١٤-١). وتأتي الحالات في العمود الذي على اليمين، و تأتي المُطلقات على طول الصف الأعلى في الجدول. ويتم تفسير الجدول وقراءته كالتالي: عندما يكون الكائن في حالة ما و يستقبل مُطلقاً محدداً، ينتقل الكائن إلى الحالة الناتجة المحددة في خلية تقاطع صف الحالة مع عمود المُطلق. وتعني الشرطة (-) أنه لا يحصل انتقال أو أن التركيبة مستحيلة. ويمكن أن تساعد معاينة الحالات والانتقالات على شكل جدول عندما تريد الإسراع، لكن لا تعتمد عليها كثيراً؛ ويمكن أن تكون تفاصيل الحالات والانتقالات أكثر تعقيداً، حينئذ يصبح العمل أسهل مع مخططات الحالة.

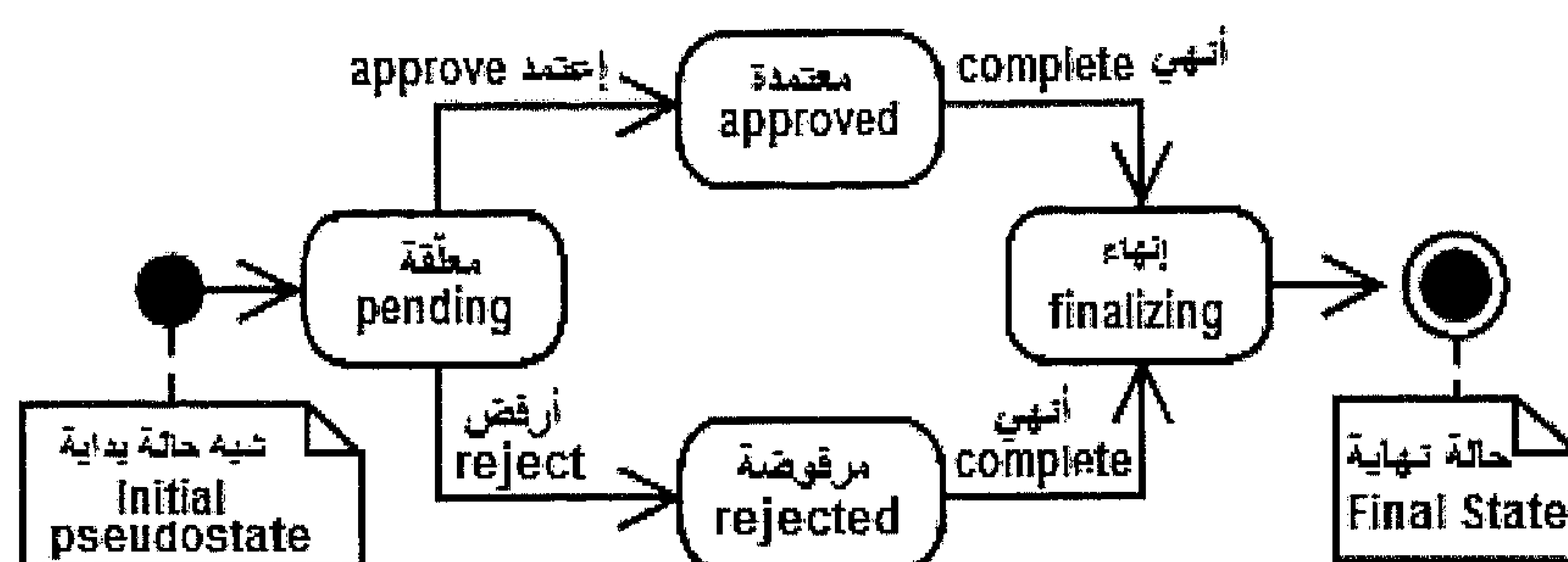
جدول رقم (١٤-١) جدول معاينة حالات وانتقالات الضوء (ليس من ترميزات UML).

الحالة / المُطلق	رفع مفتاح الضوء	إنزال مفتاح الضوء
مطفأ off	مضاء on	-
مضاء on	-	مطفأ off

عادة ما يكون لمخططات الحالة شبه حالة بداية **initial pseudostate** وحالة نهاية **final state** تحدد على التوالي نقاط البداية

والنهاية لآلة الحالة. وترسم شبه حالة البداية بواسطة دائرة مُعبّأة، وترسم حالة النهاية بواسطة دائرتَين مَركزيّتَين حيث تكون الدائرة الداخلية مُعبّأة، كما هو معروض في الشكل رقم (١٤-٣).

تعتبر شبه الحالات علامات خاصة تقوم بتوجيه حركة المرور في مخطط الحالة. كما رأينا سابقاً، وتنمذج شبه حالة البداية نقطة بداية مخطط الحالة. كما يوجد شبه حالات أخرى نراها لاحقاً في القسم "شبه الحالات المتقدمة" حيث تنمذج الانتقالات المعقدة بين الحالات. بعدما رأينا العناصر الأساسية لمخططات الحالة، دعنا ننظر في تفصيل هذه العناصر.



شكل رقم (١٤-٣) شبه حالة بداية و حالة نهاية لمخطط حالة **AccountApplication**.

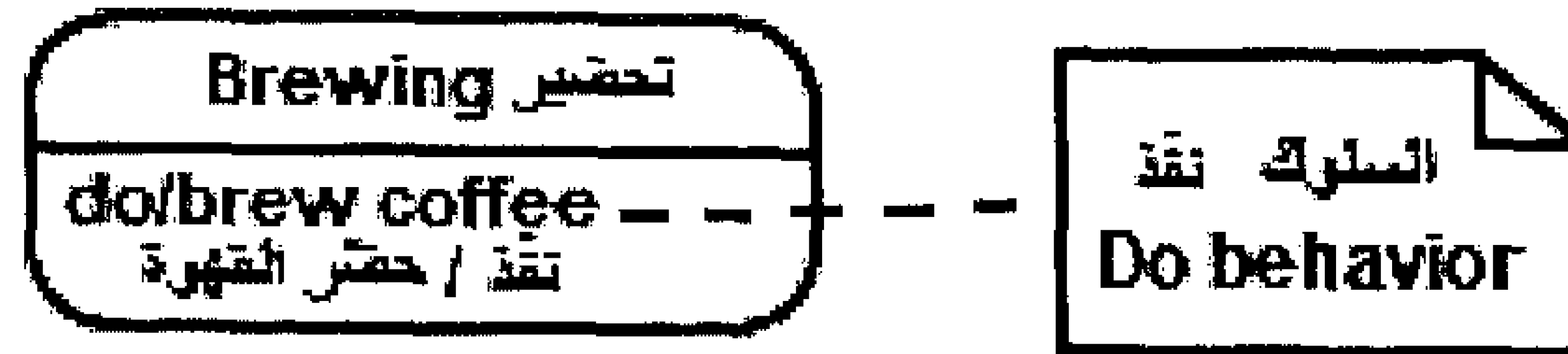
١٤-٢ الحالات States

الحالة هي وضعية وجود عند وقت محدد. ويمكن أن تكون الحالة خاصية خاملة، مثل الحالة مضاء **on** أو مطفأ **off** للكائن ضوء. ويمكن أيضاً أن تكون الحالة خاصية نشطة، أو شيئاً ما يعمله الكائن. على سبيل المثال، إن لصانعة القهوة الحالة تحضير **brewing** التي تقوم خلالها بتحضير القهوة. وترسم الحالة كمستطيل مدوّر الزوايا مع وضع اسم الحالة في وسطه، كما هو معروض في الشكل رقم (١٤-٤).

تحضير
Brewing

شكل رقم (١٤-٤) الطريقة الأكثر شيوعاً لرسم الحالة.

إذا كانت الحالة عبارة عن حالة "عمل" أو نشاط مستمر، فيمكن كتابة السلوك داخل الحالة، كما هو معروض في الشكل رقم (١٤-٥).



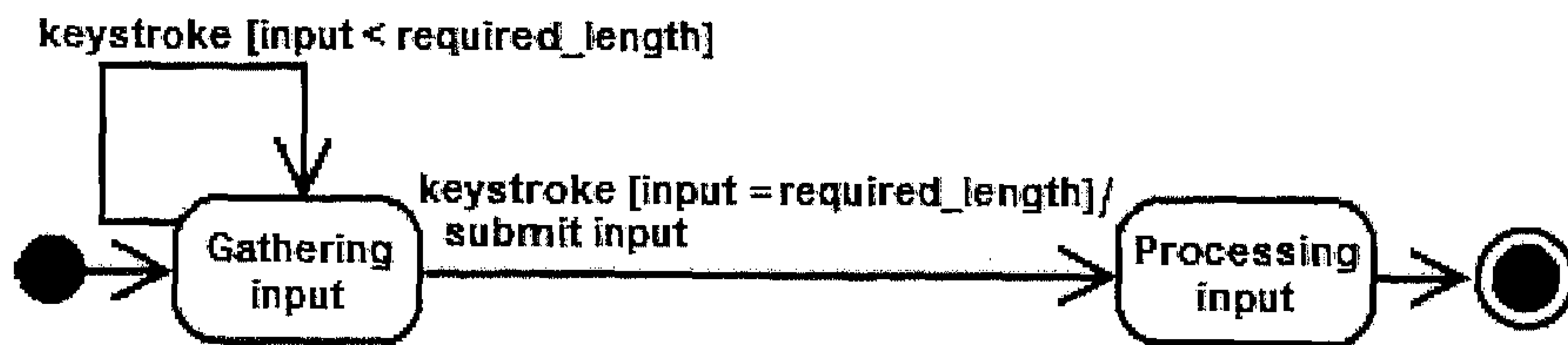
شكل رقم (١٤-٥) عرض تفاصيل السلوك لحالة "عمل".

إن السلوك نفّذ **do**، الذي يكتب بالصيغة **do/behavior**، هو سلوك يحصل ما دامت الحالة نشطة. على سبيل المثال، إن صانعة القهوة في الشكل رقم (١٤-٥) تنفّذ السلوك حضّر القهوة **brew coffee** ما دامت في الحالة تحضير **brewing**. و بشكل مماثل، ويمكن أن يكون لمشغل الأقراص المدمجة السلوك **do/read disc** (نفّذ/اقرأ القرص) ما دام في الحالة يشتغل **playing**. إن السلوك **do** إمّا أن ينتهي من تلقاء نفسه أو يُجبر على الانتهاء عندما يسبب مُطلِقاً ما الخروج من الحالة، كما تم مناقشته في القسم "الانتقالات" من هذا الفصل، سنرى طرق إضافية لعرض تفاصيل الحالة، بما في ذلك سلوكَي الدخول والخروج، وردود الأفعال على الأحداث داخل الحالة، والحالات داخل الحالات.

١٤-٣ الانتقالات Transitions

يُمثل الانتقال تغيّراً في الحالات من حالة مصدر source state إلى حالة هدف target state حيث يتم عرضه بواسطة سهم. ويكتب وصف الانتقال transition description بمحاذاة السهم، وهو يصف الظروف المسببة لحدوث تغيير الحالة.

كان لمخططات الحالة السابقة في هذا الفصل توصيفات انتقال بسيطة جداً لأنها مؤلفة من مُطلِقات فقط. على سبيل المثال، يقوم الضوء في الشكل رقم (١٤-٢) بتغيير حالته استجابة للمُطلقَين رفع المفتاح lift switch وإنزال المفتاح lower switch. ولكن يمكن أن تكون توصيفات الانتقال أكثر تعقيداً. ويتكون الترميز الكامل لتوصيفات الانتقال من الصيغة trigger [guard] / behavior، حيث إن كل العناصر اختيارية، كما هو معروض في الشكل رقم (١٤-٦). ويقوم هذا القسم بتعريف هذه العناصر، وبعد ذلك سنعرض في القسم "اختلافات الانتقال" كيفية تفاعل هذه العناصر لنمذجة أنواع مختلفة من تغييرات حالة.



شكل رقم (١٤-٦) ينمذج مخطط الحالة لمعالجة المدخلات الخصائص: المُطلق والحارس وسلوك الانتقال بمحاذاة أحد انتقالاته.

إن المُطلق trigger عبارة عن حدث قد يتسبب بحصول انتقال. في نظام معالجة مدخلات المستخدم، فإن المُطلق الضغط على مفتاح

keystroke قد يتسبب في تغيير الحالات من تجميع المدخلات **gathering input** إلى معالجة المدخلات **processing input**.



بالإضافة إلى المُطلقات، يمكن أن يتم تنشيط الانتقالات من خلال انتهاء السلوك الداخلي، كما تم مناقشته لاحقاً في هذا الفصل.

يكون الحارس **guard** عبارة عن شرط منطقي يسمح بإجراء الانتقال من عدمه. وعند وجود الحارس، يتم السماح بإجراء الانتقال إذا كانت قيمة الحارس صحيحة true، ولكن يتم اعتراض إجراء الانتقال إذا كانت قيمة الحارس خاطئة false. لنستأنف مثال مدخلات المستخدم، بعد حدوث المُطلق الضغط على مفتاح، يمكن استعمال حارس لمنع إجراء الانتقال إذا كانت المدخلات أقل من الطول المطلوب required length. وعادة ما يستعمل الحرّاس لنمذجة اعتراض إجراء الانتقال أو لنمذجة الاختيار من بين عدة انتقالات، كما تم مناقشته لاحقاً في القسم "اختلافات الانتقال".

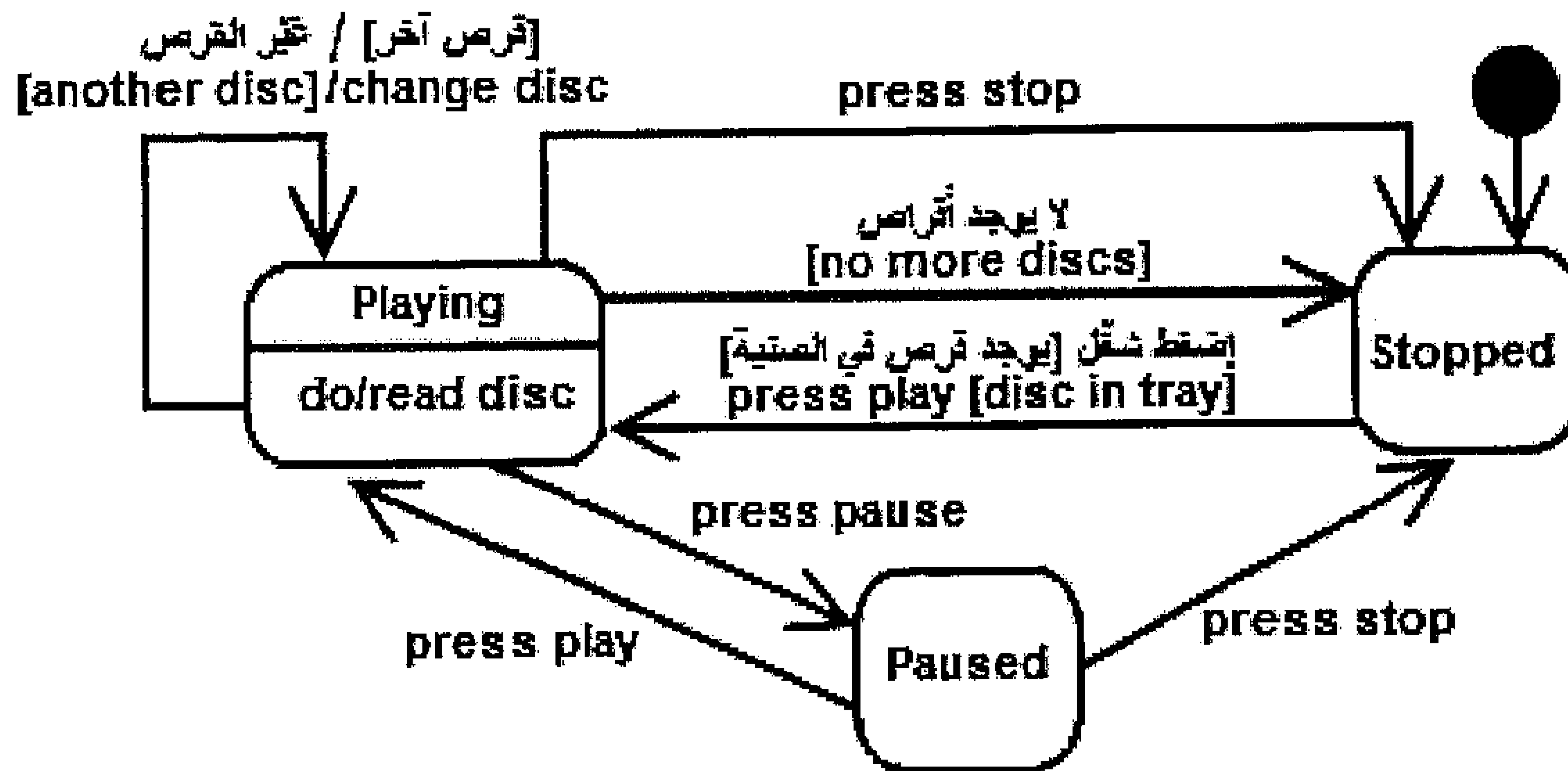
يكون سلوك **behavior** الانتقال عبارة عن نشاط غير قابل للإيقاف ويتم تنفذه بينما يحدث الانتقال (إذا تم إجراء الانتقال). على سبيل المثال، يمكن أن يتضمن سلوك الانتقال تقديم مدخلات المستخدم submit input للمعالجة بينما تتغيّر الحالات من تجميع المدخلات إلى معالجة المدخلات.

ويعرض الشكل رقم (١٤-٦) كل العناصر الثلاثة المُطلق والحارس و سلوك الانتقال. عندما يحدث الضغط على مفتاح وتكون المدخلات بالطول المطلوب، فيتم الانتقال من تجميع المدخلات إلى معالجة

المـدخلات. وأثنـاء حـدوث الانتقـال، ويـتم تطبيـق سـلوك الانتقـال تقـديم المدخلات **submit input**. يعـرض الشـكل رقـم (١٤-٦) أيضـاً أنـه يمكـن للحالة من الانتقال إلى نفسها؛ وهذا معروف بالانتقال الذاتي.

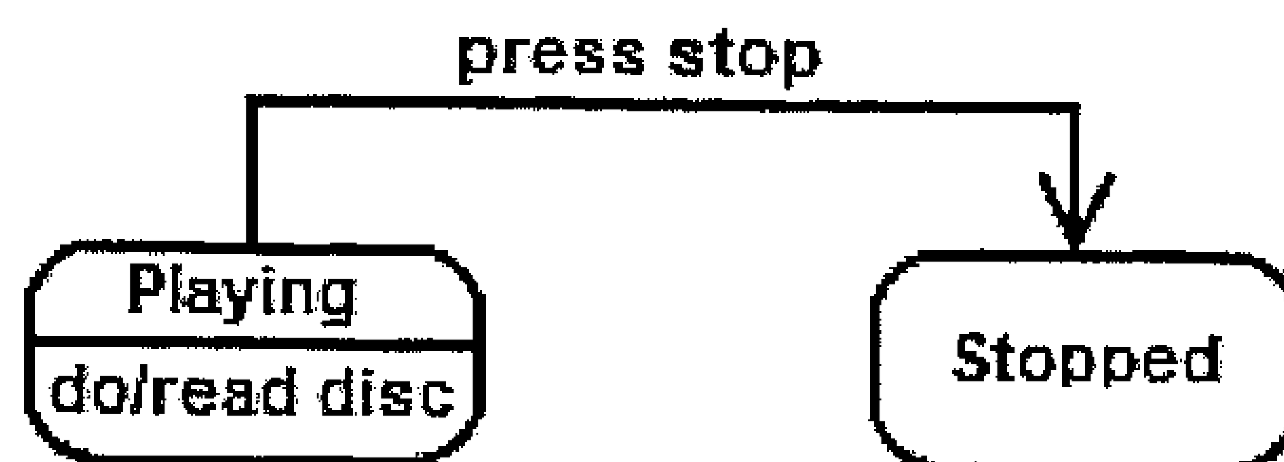
١٤-٣-١ اختلافات الانتقال Transition Variations

يعرض الشكل رقم (١٤-٧) مخطط الحالة لمشغل أقراص مدمجة. وتعرض توصيفات انتقاله تشكيلة من المُطلقات و الحرّاس، و سلوكيات الانتقـال. دعنـا نقسـم هـذا المخطـط إلى أجـزاء منفصـلة لنـدرك كيفيـة اسـتعمال تركيبـات مـن الحـرّاس و المُطلقـات لنمذجـة أنـواع مختلفـة مـن تغييرات حالة.



شكل رقم (١٤-٧) مخطط الحالة لمشغل أقراص مدمجة فيه تشكيلة توصيفات انتقال.

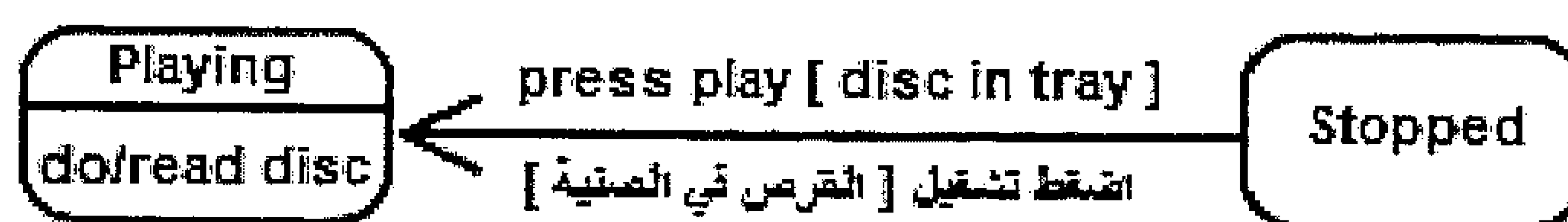
إذا تم تحـدد مُطلـق مـا مـن دون حـارس، فيـتم الانتقـال عنـد حـدوث المُطلق. يفيد هذا في نمذجة تغيير الحالـة اسـتجابة لحـدث مـا. ينتقـل مشـغل الأقـراص المدمجـة، في الشـكل رقـم (١٤-٨)، مـن الحالـة يشـتغل **Playing** إلى الحالـة متوقـف **Stopped** عنـدما يحـدث المُطلـق اضـغط إيقـاف **press stop**.



شكل رقم (١٤-٨) نوع الانتقال الأكثر شيوعاً هو الذي يقدم مُطلقاً واحداً فقط.

إذا تم تحديد مُطلق و حارس، فيتم الانتقال عندما يحدث المُطلق وتكون قيمة الحارس صحيحة، وإلا فلا يتم الانتقال. ويفيد جمع المُطلق والحارس في نمذجة إمكانية اعتراض حدوث الانتقال بالاعتماد على شرط ما. ويمكن استعمال الحرّاس أيضاً لنمذجة الاختيار في الانتقال من بين عدة انتقالات، كما سترى ذلك لاحقاً.

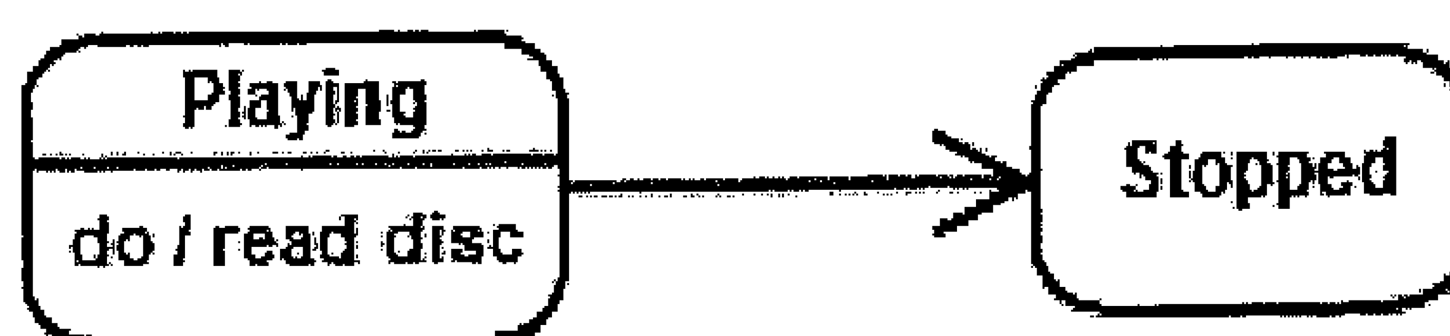
في الشكل رقم (١٤-٩)، ينتقل مشغل الأقراص المدمجة من الحالة متوقف **Stopped** إلى الحالة يشتغل **Playing** عند حدوث اضغط تشغيل **press play**، لكن إذا كان يوجد قرص في صنية المشغل فقط.



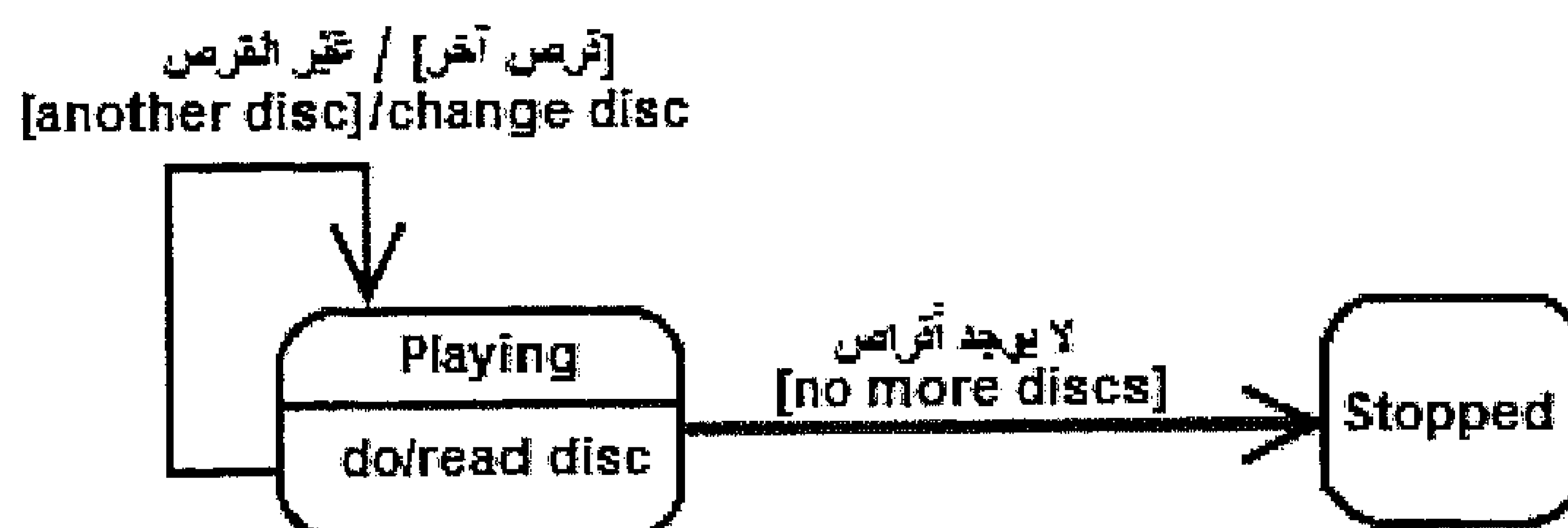
شكل رقم (١٤-٩) سيعترض الحارس إجراء الانتقال إذا كانت قيمته خطأ.

إذا لم يتم تحديد لا مُطلق و لا حارس، فيتم الانتقال مباشرة بعد انتهاء السلوك الداخلي للحالة المصدرية (إذا كان السلوك موجوداً). ويفيد هذا في نمذجة انتقال سببه انتهاء السلوك الداخلي. يعرض الشكل رقم (١٤-١٠) انتقالاً من دون مُطلق ولا حارس، حيث يقود من الحالة يشتغل **Playing** إلى الحالة متوقف **Stopped**، مما يعني أن مشغل الأقراص المدمجة ينتقل إلى الحالة متوقف عندما ينهي قراءة القرص. (إن هذا الانتقال غير ظاهر في مخطط الحالة الكامل لمشغل الأقراص المدمجة

المعروض في الشكل رقم (١٤-٧)، ولكنه معروض في الشكل رقم (١٤-١١) لتوضيح الانتقالات من دون مُطلق).



شكل رقم (١٤-١٠) سبب الانتقال هنا هو انتهاء السلوك الداخلي.



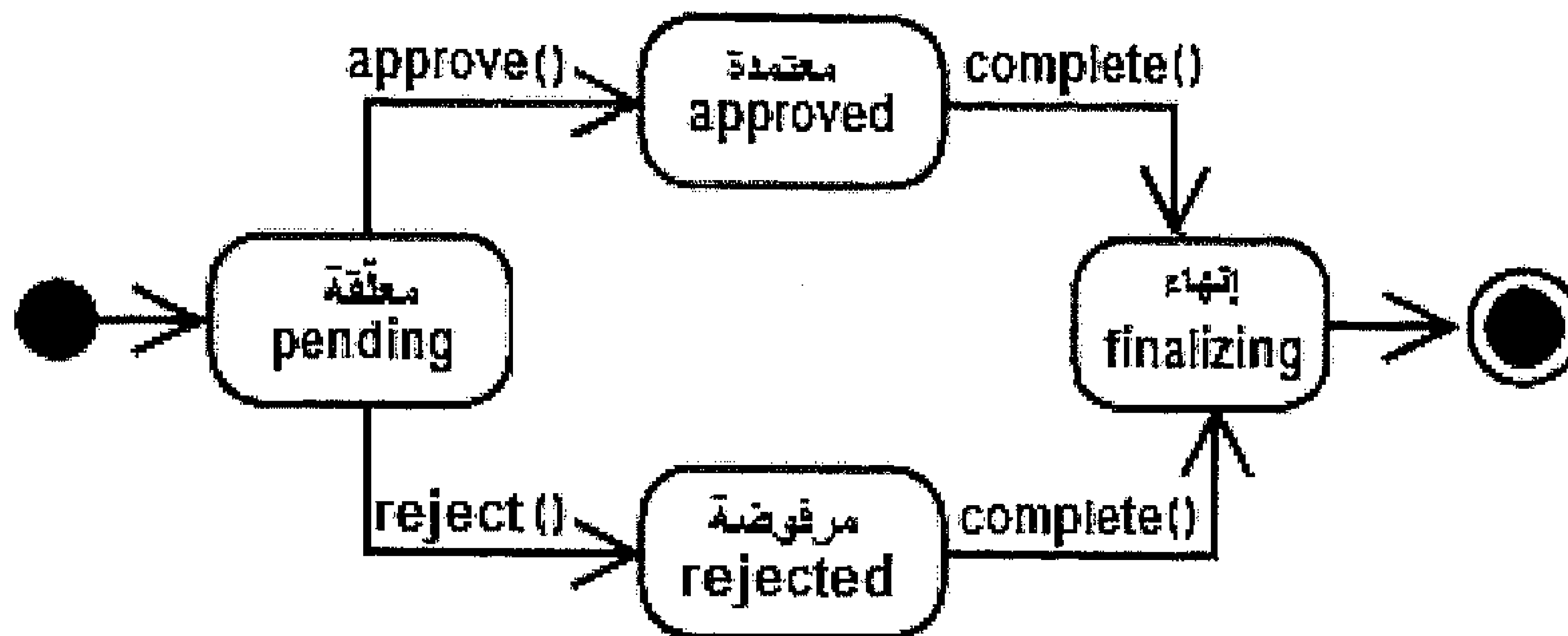
شكل رقم (١٤-١١) استعمال الحرّاس لنمذجة الاختيار من بين عدة المسارات.

يعرض الشكل رقم (١٤-٩) استعمال الحرّاس لاعتراض إجراء الانتقال. ويمكن استعمال الحرّاس أيضاً لعرض اختيار انتقال من بين عدة انتقالات: إن الانتقال الذي تكون قيمة حارسه صحيحة هو الذي يتم اختياره. وفي الشكل رقم (١٤-١١)، بعد انتهاء مشغل الأقراص المدمجة من قراءة القرص، إما أن ينتقل إلى الحالة متوقف **Stopped** إذا لم يوجد المزيد من الأقراص، وإما أن يرجع إلى الحالة يشتغل **Playing** إذا كان هناك المزيد من الأقراص. ويجب الانتباه لحالة وجود المزيد من الأقراص حيث يتضمن الانتقال سلوك الانتقال غيّر القرص **change disc**.

يمكن استعمال شبه حالة الاختيار كعرض بديل للخيارات، والذي ستتم مناقشته لاحقاً في القسم "شبه الحالات المتقدمة".

١٤-٤ الحالات في البرامج States in Software

إذا كنت مطوّر برامج، فقد تتفاجأ عندما تحتاج إلى نمذجة تشغيل مشغل الأقراص المدمجة أو صانع القهوة. في البرامج، يقوم مخطط الحالة بنمذجة دورة حياة كائن أو الحالات التي يمرّ بها أثناء حياته المتوقعة. يعرض الشكل رقم (١٤-١٢) دورة حياة كائن حساب **AccountApplication** كما ينتقل من الحالة مُعلّق **pending** إلى الحالة مُعتمد **approved** أو مرفوض **rejected**، و من ثم للحالة إنهاء **finalizing**.

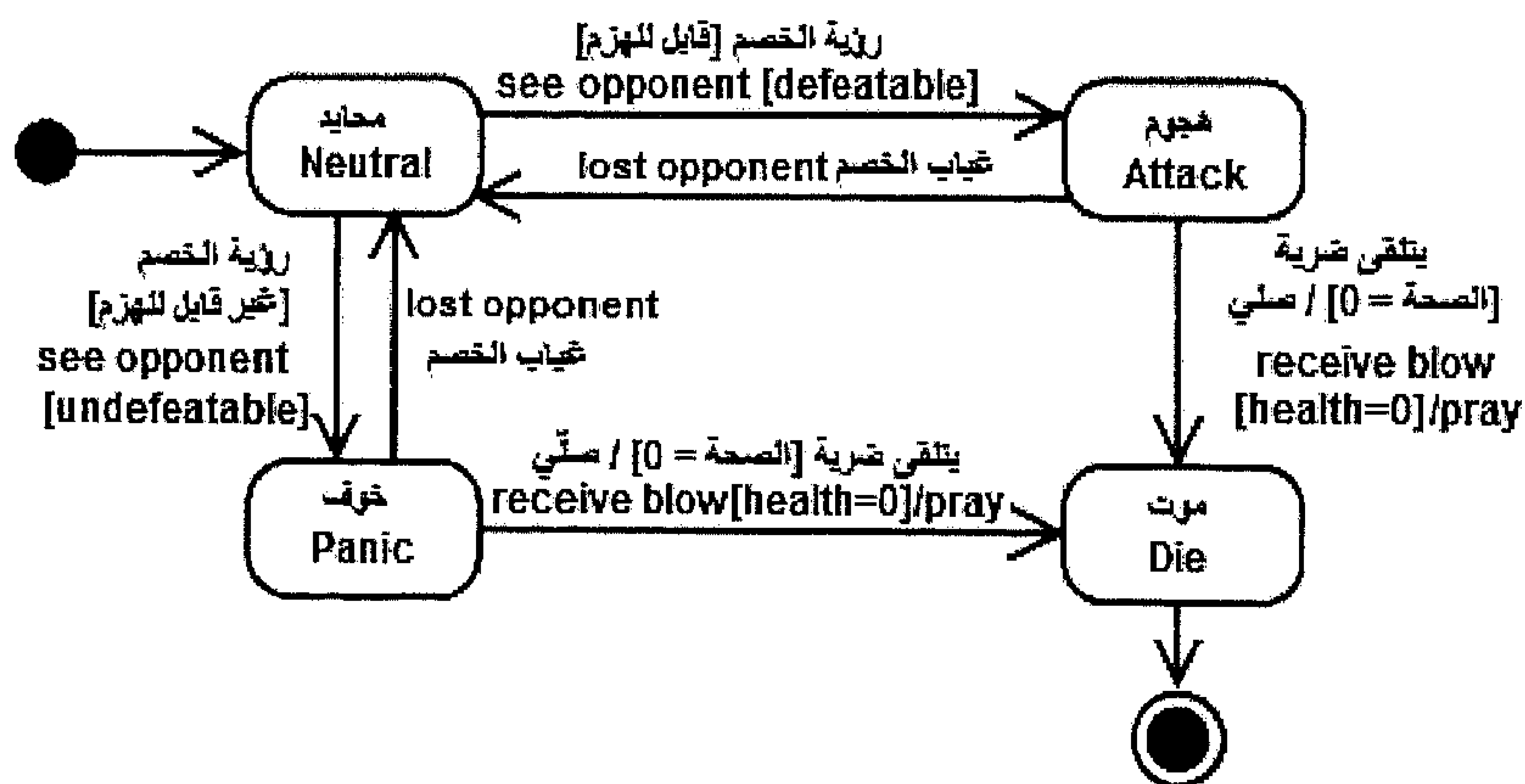


شكل رقم (١٤-١٢) دورة حياة كائن حساب تطبيقي **AccountApplication**.

تفيد مخططات الحالة في نمذجة كائن ما يتصرّف بشكل مختلف بالاعتماد على حالته. ولنأخذ بالاعتبار كائن **AccountApplication**، إن استدعاء الطريقة أنهي **complete()** عندما يكون الكائن في الحالة معلّق **pending**، ليس له معنى إذا كانت الحالة إنهاء **finalizing** تنفذ سلوك لفّ القرص، مثل إنشاء حساب مدونة إذا تم الموافقة عليه (فهو يتطلب أولاً معرفة إذا تم الموافقة على الطلب). وتشكل مخططات الحالة وسيلة فعّالة لجعل هذه المعلومات صريحة و واضحة.

إذا كان للكائن دورة حياة بسيطة، فليس من المجدي نمذجة دورة حياته باستعمال مخطط الحالة. على سبيل المثال، إن الكائن معلومات الاتصال **ContactInformation**، الذي يخزن معلومات الاتصال بكاتب ما، لا يغيّر حالاته عن كونه مُنشأ أو مُهدّم، و هذا لا يبرر استعمال مخطط حالة له.

يتم استعمال مخطط الحالة أيضاً بكثرة في بعض البرامج المتخصصة، مثل ألعاب الرماية من منظور الشخص الأول First Person Shooter (FPS). في هذه الألعاب، يتم استعمال مخطط الحالة لنمذجة حالات شخصية اللعبة. على سبيل المثال، يمكن أن يكون للعبة ما (مثل لعبة الصيّاد **troll**) الحالات محايد **Neutral** و هجوم **Attack** وخوف **Panic**، وموت **Die**، كما هو معروض في الشكل (١٤-١٣). عندما يكون الصياد في الحالة هجوم، يكون يؤدّي سلوكاً ما، مثل استلال السيف أو الانقضاض على الخصم. تتسبب المُطلقات بتغيير الحالة يشمل رؤية الخصم see opponent أو تلقي ضربة receive blow من الخصم.



شكل رقم (١٤-١٣) نمذجة مخطط الحالة لصياد في لعبة **FPS**؛ يتم تحديد سلوك الصياد من خلال حالته.



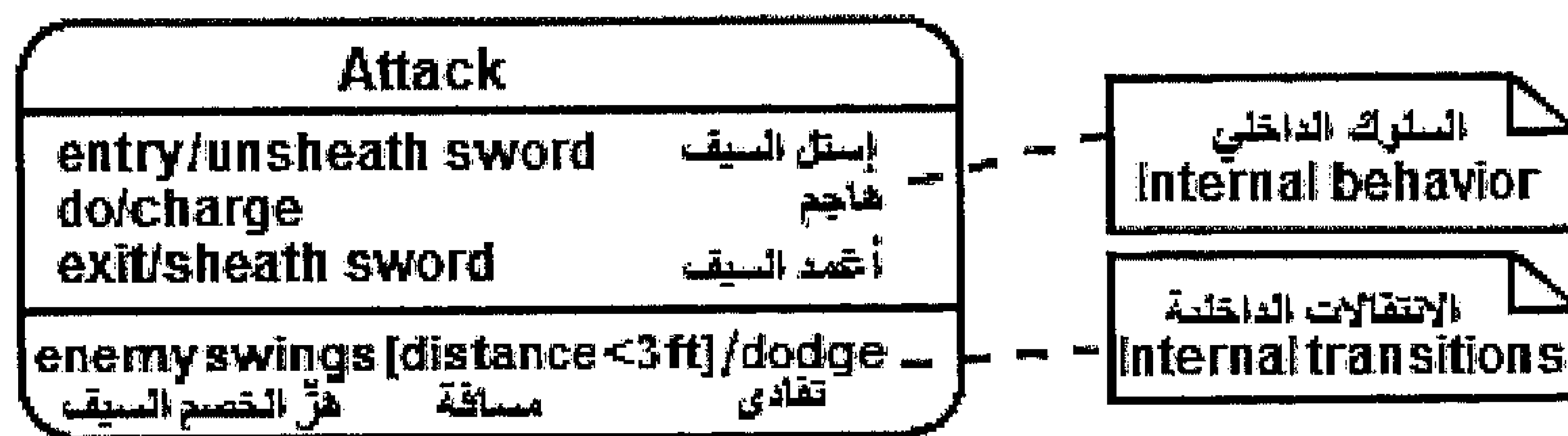
إذا كنت تتساءل عن شفرة برمجة حالات كائن ما، يمكن أن يكون للصنف AccountApplication الخاصية حالة status حيث تكون قيمها المحتملة هي الحالات التي في الشكل رقم (١٤-١٢). وتحدث الانتقالات عندما يتم استدعاء الطرق من خلال كائن AccountApplication لتطبق عليه. انظر إلى الفصل الرابع لمراجعة كيفية أسر حالة الكائن في خصائصه.

١٤-٥ السلوك المتقدم للحالة

Advanced State Behavior

لقد رأيت الأساليب الأكثر شيوعاً لنمذجة الحالات. ويعرض هذا القسم كيف تنمذج التفاصيل الإضافية لحالة محددة، بما في ذلك سلوك الدخول، سلوك الخروج، وردود الفعل على الأحداث أثناء الوجود في حالة محددة.

ويعرض الشكل رقم (١٤-١٤) الترميز المفصّل للحالة الممثل بمستطيل كبير مدوّر الزوايا ذي عدة مقصورات منفصلة من أجل السلوك الداخلي والانتقالات الداخلية.



شكل رقم (١٤-١٤) السلوك الداخلي والانتقالات الداخلية للحالة هجوم Attack.

١٤-٥-١ السلوك الداخلي Internal Behavior

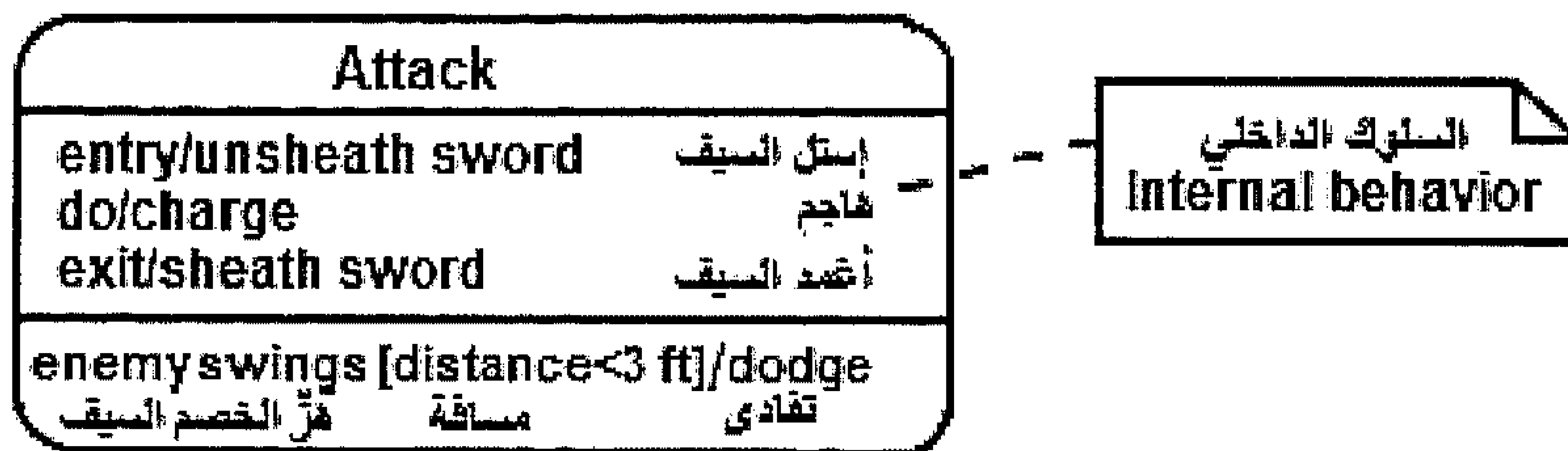
السلوك الداخلي عبارة عن أي سلوك يحدث ما دام الكائن موجوداً في حالة ما. لقد رأينا سابقاً السلوك نفّذ do، الذي يمثل سلوكاً

متواصلاً ما دامت الحالة نشطة. إن السلوك الداخلي هو مفهوم أكثر عمومية حيث يتضمّن أيضاً السلوك دخول، والسلوك خروج.

تتم كتابة السلوك الداخلي بالصيغة label / behavior. يحدد العنصر عنوان label متى سيتم تنفيذ السلوك، أي الأحداث أو الظروف المسببة للسلوك. هناك ثلاث عناوين خاصّة: دخول **entry**، خروج **exit** ونفّذ **do**.

يحدث السلوك دخول بعدما تصبح الحالة نشطة مباشرة، و تتم كتابته بالصيغة entry/behavior. ويحدث السلوك خروج قبل أن تصبح الحالة غير نشطة مباشرة و تتم كتابته بالصيغة exit/behavior.

في الشكل رقم (١٤-١٥)، عندنا استلال السيف **unshealth sword** مثالاً لسلوك دخول و غمد السيف **shealth sword** مثالاً لسلوك خروج. بخلاف السلوك نفّذ **do**، لا يمكن إيقاف أو اعتراض سلوكي الدخول و الخروج.



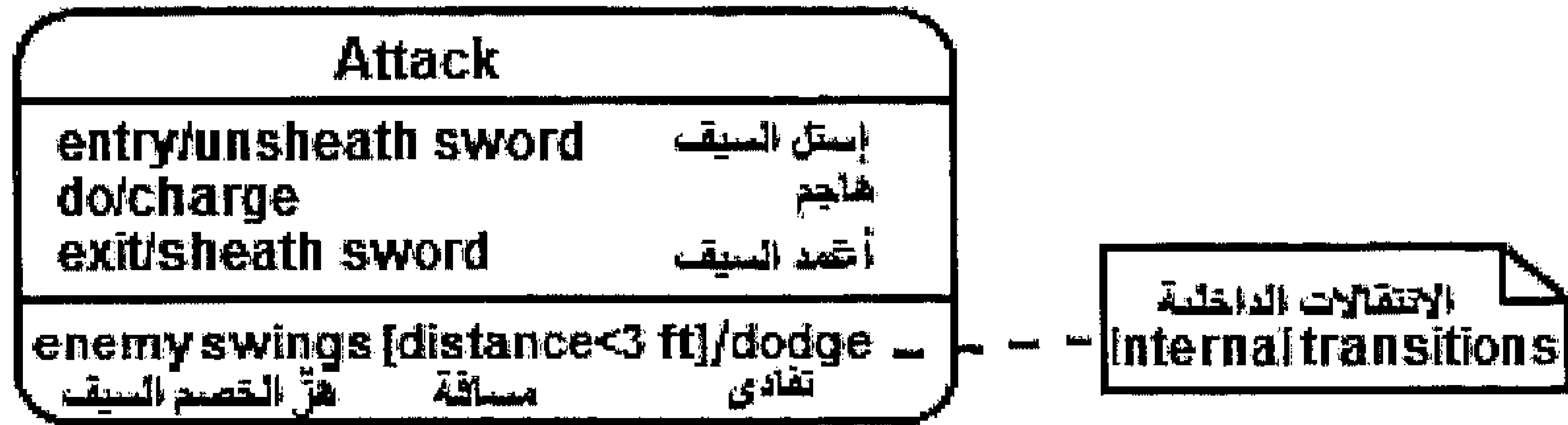
شكل رقم (١٤-١٥) تعرض المقصورة الوسطى السلوك الداخلي.

١٤-٥-٢ الانتقالات الداخلية Internal Transitions

إن الانتقال الداخلي عبارة عن انتقال يتسبب بردة فعل داخل الحالة، ولكنه لا يتسبب بتغيير حالات الكائن. ويختلف الانتقال الداخلي

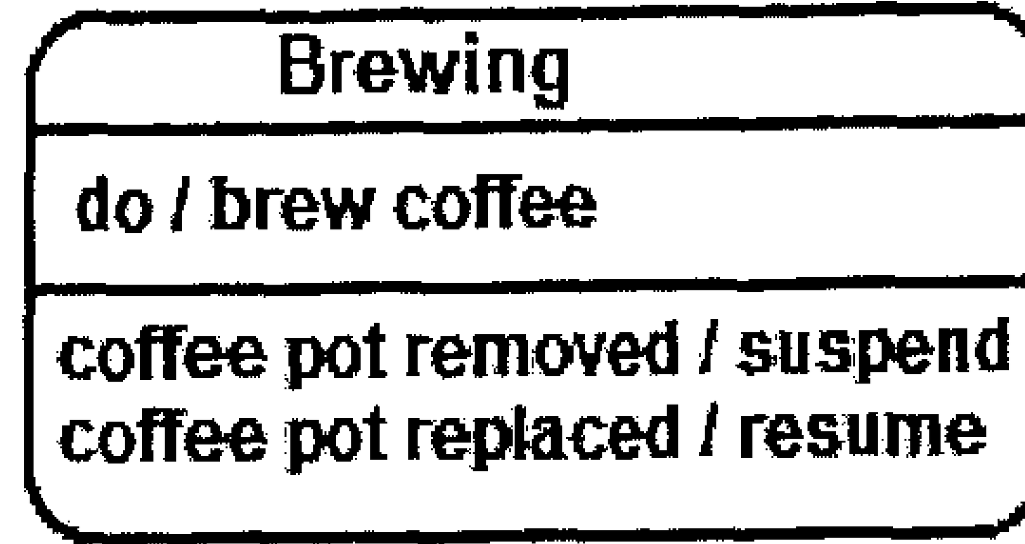
عـن الانتقـال الـذاتي لأن الانتقـالات الذاتيـة تتسـبب في حـدوث سـلوكي الـدخول والخـروج بينمـا لا تتسـبب بـذلك الانتقـالات الداخليـة، انظـر إلى الشكل رقم (١٤-١١).

تُكتب الانتقـالات الداخليـة بالصـيغة **trigger [guard] / behavior** (المُطلـق / [الحـارس] السـلوك) حيـث يـتم إدراجهـا داخـل الحالـة. وفي الشـكل رقم (١٤-١٦)، إن للحالة هجوم **Attack** انتقالاً داخلياً: عندما يهـز الخصم سيفه ويكون على مسافة أقل من ثلاثة أقدام مـن الصـياد، فيقـوم الصياد بتفاديه.



شكل رقم (١٤-١٦) تعرض المقصورة السفلية الانتقالات الداخلية.

قـم باسـتخدام الانتقـالات الداخليـة في نمذجـة ردود الأفعـال علـى الأحداث التي لا تتسبب بتغيير الحـالات. على سبيل المثال، يمكـن استخدام الانتقالات الداخليـة مـع صـانعة القهـوة ذات الخـاصـية توقـف وأخـدم، حيـث إنها تقوم بالتوقف عن توزيع القهوة عنـد إزالـة إبريـق القهـوة coofee pot مـن دون تغـيير الحالـة تحضـير **Brewing**، وتقـوم باسـتئناف **resume** التحضـير عنـد إعـادة وضـع **replace** إبريـق القهـوة مكانـه، كمـا هـو معـروض في الشـكل رقم(١٤-١٧).

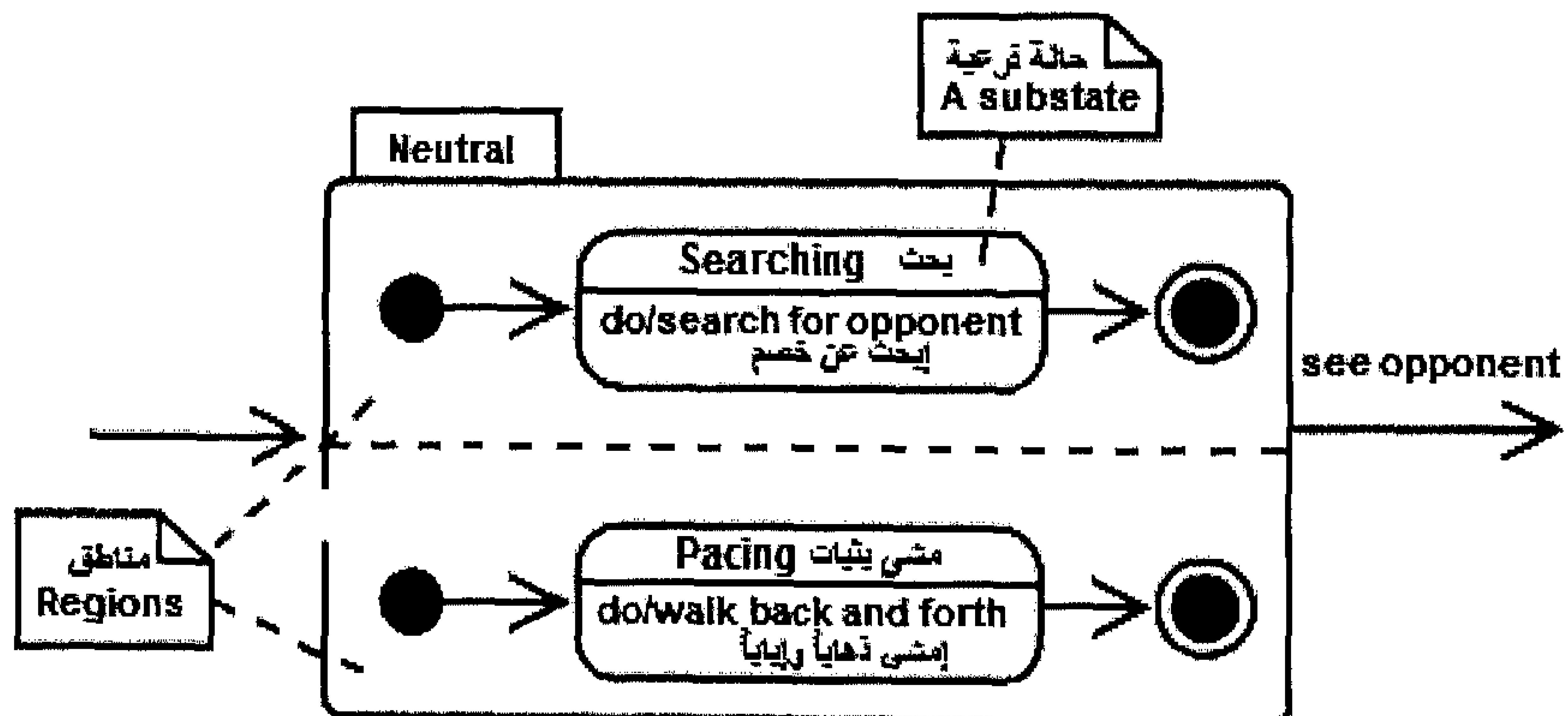


شكل رقم (١٤-١٧) ينمذج الانتقال الداخلي ردة الفعل بينما يبقى في نفس الحالة.

١٤-٦ الحالات المركبة Composite States

هناك اختلاف رئيسي بين مخططات الحالة في لغة النمذجة الموحدة و مخططات الحالة الأخرى، وقد يكون مألوفاً لديك، وهو أن لغة النمذجة الموحدة تسمح بالحالات المتزامنة، أو بالتواجد في عدة حالات بنفس الوقت. إن الحالات المركبة هي التي تجعل ذلك ممكناً.

افترض أن الصياد هو في الحالة محايد **Neutral** يقوم بعمل أمرين في نفس الوقت: يبحث **Searching** ويركض **Pacing**. يمكنك نمذجة هاتيْن الحالتيْن المتزامنتَين باستعمال الحالة المركبة، كما هو معروض في الشكل رقم (١٤-١٨).



شكل رقم (١٤-١٨) تحتوي الحالات المركبة على مخطط حالة أو أكثر؛ إذا كانت تحتوي على أكثر من مخطط حالة فتنفذ مخططات الحالة بشكل متواز.

تكون الحالة المركبة عبارة عن حالة تحتوي على مخطط حالة واحد أو أكثر. وكل مخطط ينتمي إلى منطقة region، حيث يتم تقسيم المناطق بواسطة خط منقط. وتتم الإشارة إلى الحالة التي في منطقة ما على أنها حالة فرعية **substate** من الحالة المركبة.

وتعمل الحالات المركبة كالتالي: عندما تصبح الحالة المركبة نشطة، تصبح شبه حالة البداية لكل منطقة نشطة وتبدأ مخططات الحالة التي بداخلها بالتنفيذ. ويتم اعتراض تلك المخططات إذا حدث مُطلق على الحالة المركبة. في الشكل رقم (١٤-١٨)، يتم إيقاف الحالات الفرعية عندما يحدث المُطلق رؤية خصم **see opponent** على الحالة المركبة.

إذا كان بالإمكان تنفيذ سلوك الحالات الفرعية حتى نهايته، فتنتهي الحالة المركبة عند انتهاء كل مخططات الحالة للمناطق.

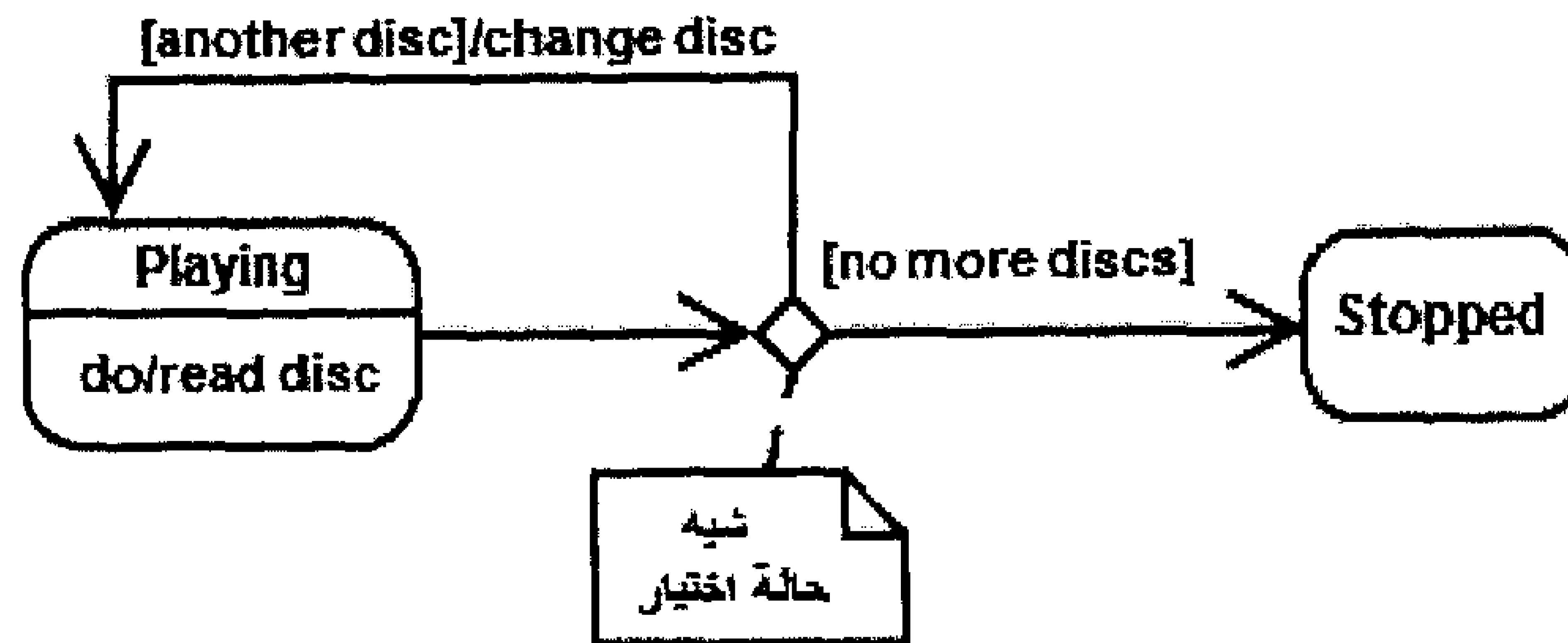
١٤-٧ شبه الحالات المتقدمة

Advanced Pseudostates

لقد رأينا سابقا شبه حالات البداية التي تحدد بداية مخطط الحالة. وهناك شبه حالات إضافية تفيد في توجيه تدفق حركة المرور بين الحالات.

تستعمل شبه حالة الاختيار **choice pseudostate** للتأكيد على وجود شرط ما يقوم بتحديد الانتقال الذي يجب إتباعه. لكل انتقال يخرج من شبه حالة الاختيار حارس يتحدد من خلاله الانتقال الذي سيتم اتباعه. في الشكل رقم (١٤-١٩)، سيرجع مشغل الأقراص المدمجة إلى الحالة يشتغل **Playing** إذا كان يوجد قرص إضافي آخر أو سيذهب إلى الحالة متوقف **Stopped** إذا لم يكن يوجد المزيد من الأقراص. لاحظ أنّ هذه

طريقة بديلة و أكثر وضوحاً لنمذجة الانتقال الاختياري في الشكل رقم (١٤-١١).



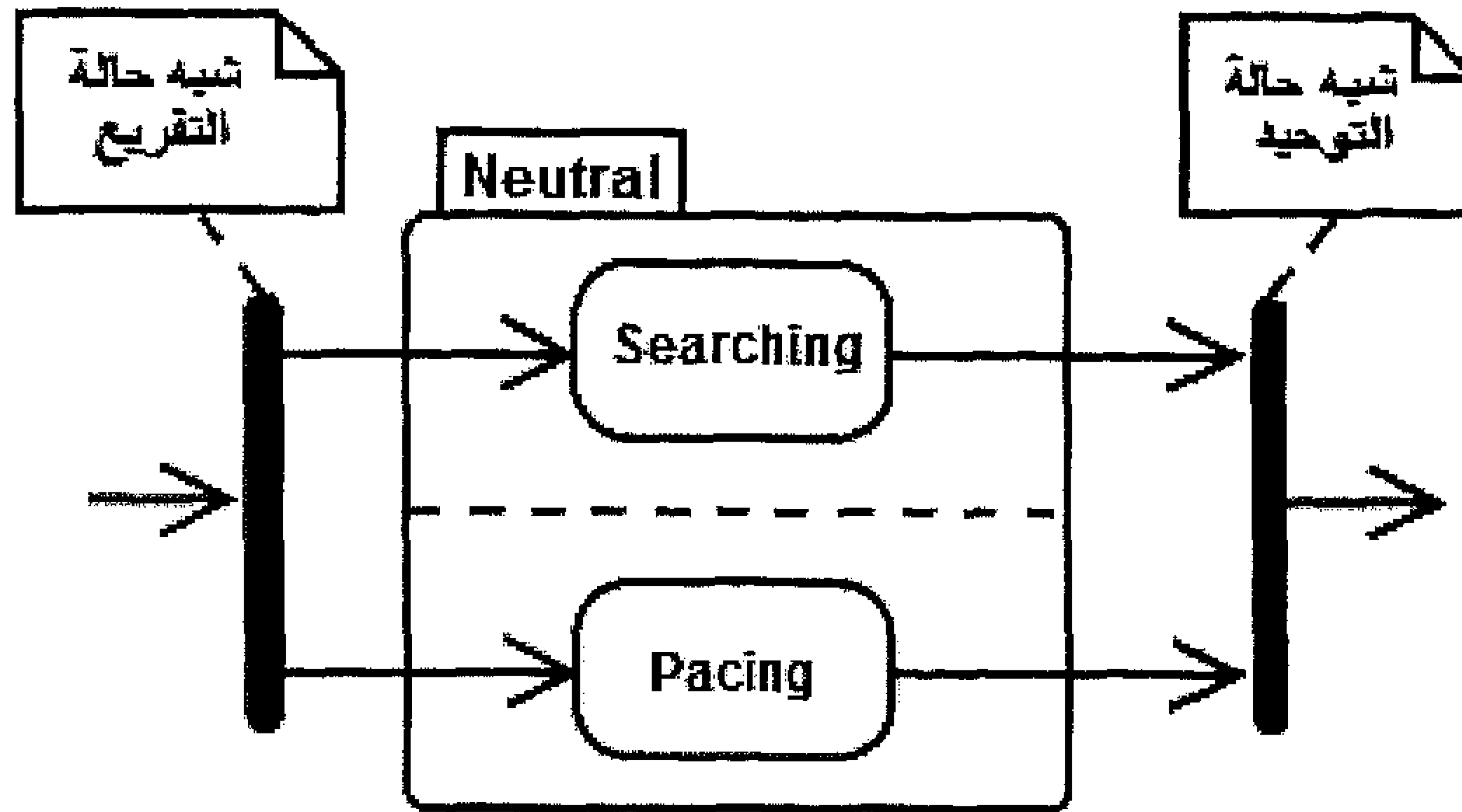
شكل رقم (١٤-١٩) يعتمد المسار المتّبع بعد الاختيار على قيمة الحارس.



يجب على حارس واحد على الأقل يأتي بعد شبه حالة الاختيار أن تكون قيمته صحيحة حتى يكون النموذج مركباً بشكل جيد. وإذا كانت قيمة أكثر من حارس واحد بعد الاختيار صحيحة، فيتم اختيار واحد منهم بشكل اعتباطي. إذا لم يكن لهذه الحالة معنى بالنسبة للنموذج، فيشير ذلك إلى ضرورة إعادة تعريف الحرّاس كي تكون قيمة واحد منهم بالضبط صحيحة في كل مرة.

وتقوم شبه حالة التفريع fork و شبه حالة التوحيد join بعرض التفرع إلى حالات متوازية ثم توحيدها من جديد. على سبيل المثال، في الشكل رقم (١٤-٢٠)، تفرق شبه حالة التفريع الانتقال القادم إلى انتقالَين، وذلك للسماح للصياد بأن يبحث **Searching** و يمشي بثبات Pacing بشكل متزامن. ثم تقوم شبه حالة التوحيد بتوحيد الانتقالَين القادمَين إلى انتقال خارج واحد.

يعتبر الشكل رقم (١٤-٢٠) وسيلة بديلة لنمذجة الشكل (١٤-١٨). في الشكل رقم (١٤-١٨)، يكون التفريع و التوحيد ضمنياً خلال عرض شبه حالات البداية و النهاية.



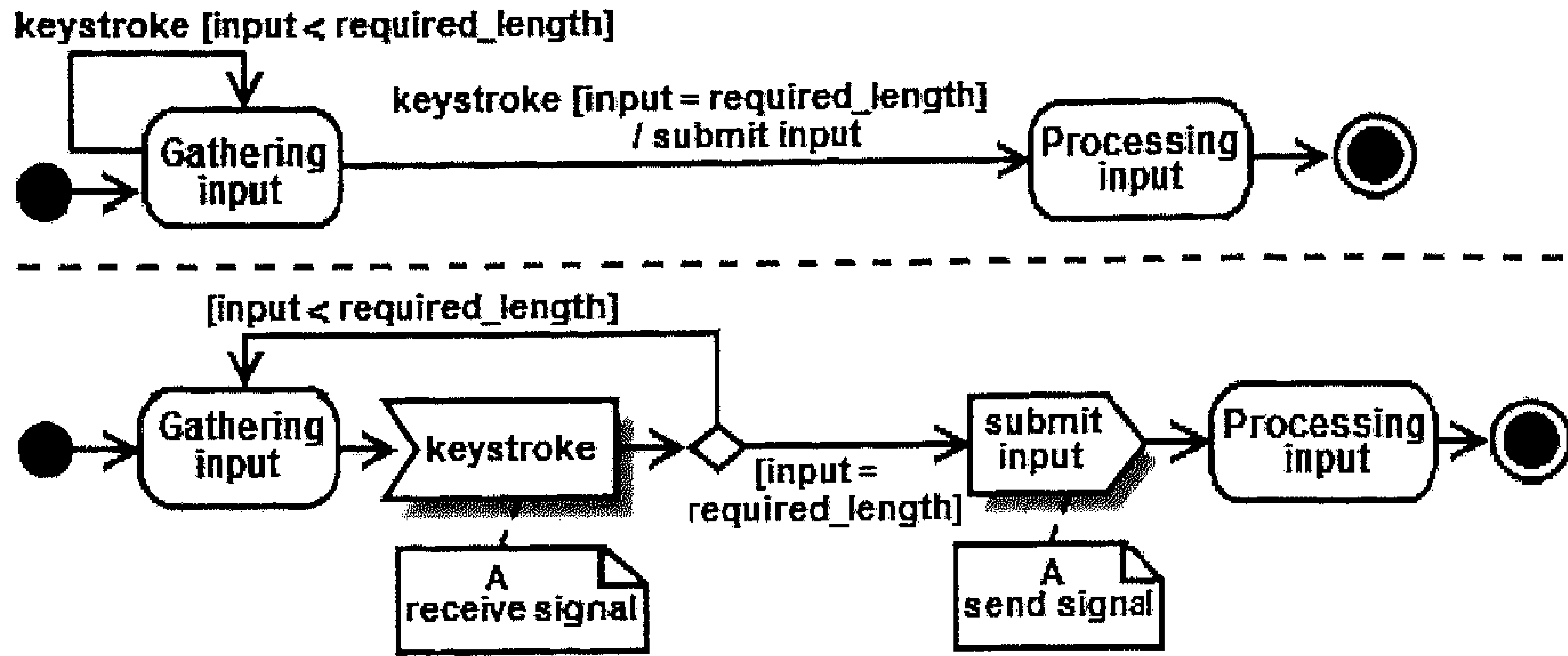
شكل رقم (١٤-٢٠) يبين التفريع والتوحيد الحالات المتزامنة.

١٤-٨ الإشارات Signals

يمكن استعمال أيقونات خاصّة للانتقالات من أجل لفت النظر إلى الانتقالات وسلوكها. يُدعى هذا بالمنظور انتقالي التوجه **transition-oriented view**.

في هذا المنظور، يتم تمثيل المُطلق باستعمال أيقونة استلام إشارة، و يتم تمثيل سلوك الانتقال باستعمال أيقونة إرسال إشارة. ويعرض الشكل رقم (١٤-٢١) كيف يمكن رسم الشكل رقم (١٤-٦) باستعمال هذا الترميز البديل. وهو يستعمل أيضاً شبه حالات الاختيار المُقدمة سابقاً في القسم "شبه الحالات المتقدّمة".

إن الهدف الرئيس لهذا الترميز هو التأكيد على إرسال الإشارات واستلامها بشكل تصويري. بالرغم من تعبير كلا المخططَين عن نفس الأمر، إنما تركز النسخة التي تستعمل أيقونات الإشارة على الانتقالات مما يجعل المخطط أكثر مقروئية.

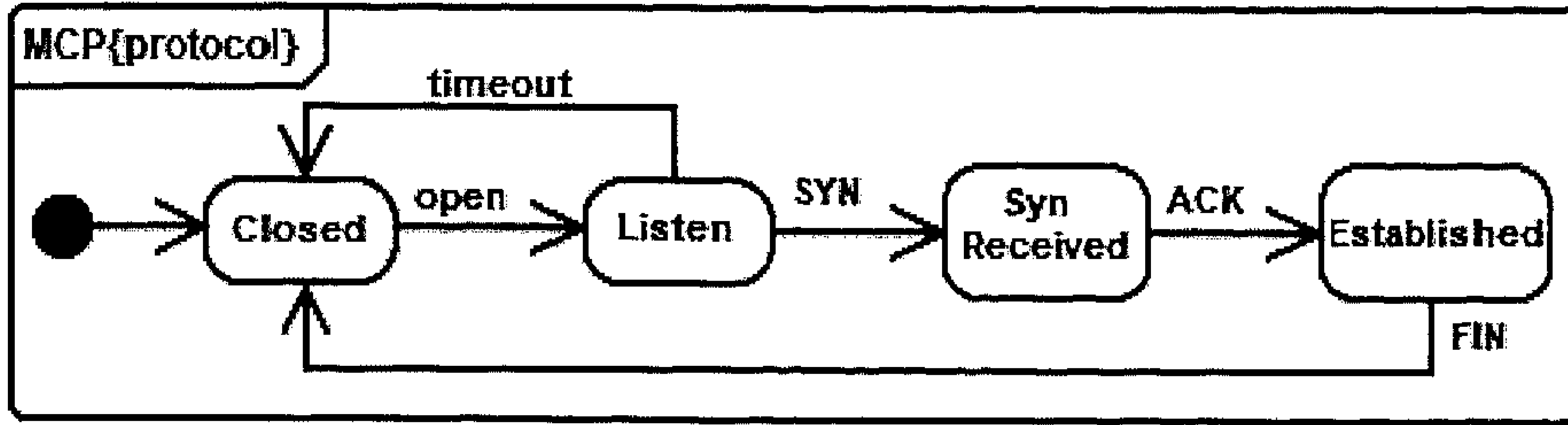


شكل رقم (١٤-٢١) يرسم المخطط السفلي الانتقالات وسلوكها كاستلام إشارات وإرسالها.

١٤-٩ آلات حالة البروتوكول

Protocol State Machines

تعتبر آلات حالة البروتوكول نوعاً خاصاً من آلات الحالة التي تركز على كيفية عمل البروتوكول، مثل بروتوكول الاتصالات (أي بروتوكول التحكم بالنقل Transmission Control Protocol – TCP). إن الاختلاف الرئيس بين آلات حالة البروتوكول و آلات الحالة السلوكية، التي ركّزنا عليها سابقاً، هو عدم عرض آلات حالة البروتوكول سلوكاً بمحاذاة الانتقالات أو داخل الحالات. وبدلاً من ذلك، فهي تركز على عرض سلسلة شرعية من الأحداث والحالات الناتجة عنها. ويتم رسم آلات حالة البروتوكول داخل مستطيل له عروة مع وضع اسم آلة الحالة داخل العروة وكتابة التعبير **{protocol}** بعده، كما هو معروض في الشكل رقم (١٤-٢٢).



شكل رقم (١٤-٢٢) بروتوكول آلة حالة يُنمذج جانب المستلم لبروتوكول اتصالات مُبسّط يدعى بروتوكول اتصالاتي (My Communication Protocol – MCP).

بما أن آلات حالة البروتوكول لا تعرض السلوك، فلا يمكن نمذجة ما يقوم بعمله النظام كاستجابة لأمر ما (على سبيل المثال، عند إرسال إقرارات بالوصول للنظام). لكن يمكن أن يفيد هذا في عرض كيفية العمل مع كائن أو نظام ما، مثل تحديد بروتوكول اتصالات أو سلسلة استدعاءات متوقعة لعمليات الكائن.

١٤-١٠ ما هي الخطوة التالية؟

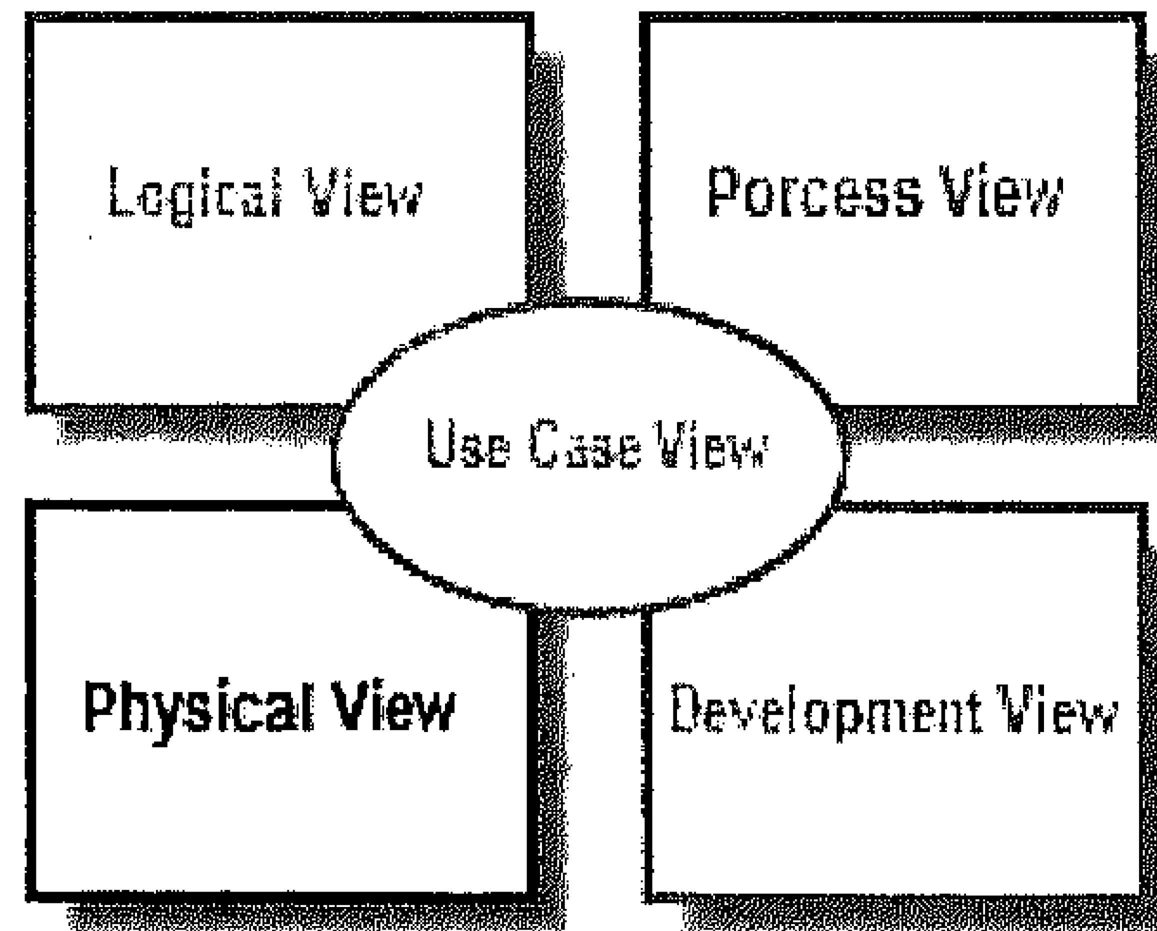
تقوم مخططات الحالة بعرض حالات الكائن و المُطلقات المسببة لتغيير حالاته. وإذا كنت مهتماً بنمذجة تغيّر حالة كائن في سياق تدفق عمل ما، فانظر إلى مخططات النشاط المغطاة في الفصل الثالث. وإذا أردت عرض التوقيت المرتبط بتغييرات الحالة، فمن الجدير مراجعة أيضاً مخططات التوقيت المغطاة في الفصل التاسع.

نمذجة النظام المنشور: مخططات النشر

MODELING YOUR DEPLOYED SYSTEM: DEPLOYMENT DIAGRAMS

إذا كنت قد قمت بتطبيق تقنيات لغة النمذجة الموحدة المعروضة في الفصول السابقة في هذا الكتاب، فتكون قد رأيت كل منظورات النظام باستثناء منظور واحد فقط. وهذا الجزء الناقص هو المنظور المادي **physical**. ويهتم المنظور المادي بعناصر النظام المادية، مثل ملفات تنفيذ البرامج و الأجهزة التي تُنفّذ عليها.

وتعرض مخططات النشر الخاصة بلغة النمذجة الموحدة المنظور المادي للنظام، وذلك بإعلان ولادة البرنامج فعلياً من خلال عرض كيفية تنصيبه على الأجهزة و كيفية التواصل مع الأجزاء، انظر الشكل رقم (١٥-١).



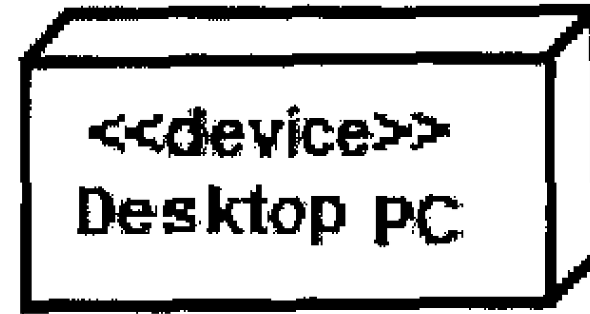
شكل رقم (١٥-١) تركز مخططات النشر على المنظور المادي للنظام.



يمكن أن تعني كلمة نظام أشياء مختلفة لعدة أشخاص؛ في سياق مخططات النشر، فهي تعني البرامج التي نقوم بإنشائها والأجهزة والبرمجيات التي تسمح بتنفيذها.

١٥-١ نشر نظام بسيط
Deploying a Simple System

دعنا نبدأ بعرض مخطط نشر لنظام بسيط جداً. في أبسط الحالات، سيتم تسليم البرنامج كملف تنفيذي يتم تنصيبه على حاسب واحد. ويتم استعمال العُقَد **nodes** لتمثيل أجهزة الحاسب، كما هو معروض في الشكل رقم (١٥-٢).



شكل رقم (١٥-٢) استعمل العُقد لتمثيل الأجهزة في النظام.

يحتوي هذا النظام على جهاز واحد بسيط (حاسب شخصي مكتبي Desktop PC) حيث تمت عنونته باستعمال الحاشية <<device>> لتحديد أنها عقدة جهاز.

نحتاج الآن إلى نمذجة البرنامج الذي يشتغل على الأجهزة. ويعرض الشكل رقم (١٥-٣) برنامجاً اصطناعياً بسيطاً software artifact (انظر لاحقا "البرمجيات المنشورة: الأدوات الاصطناعية"). يتكون هذا البرنامج في حالتنا من مجرد ملف من النوع JAR اسمه 3dpacman.jar (ملف من النوع JAR أي أن امتداد اسمه يكون JAR)، ويحتوي على تطبيق ثلاثي الأبعاد للعبة باكمان 3D-Pacman.

<<artifact>>
3dpacman.jar

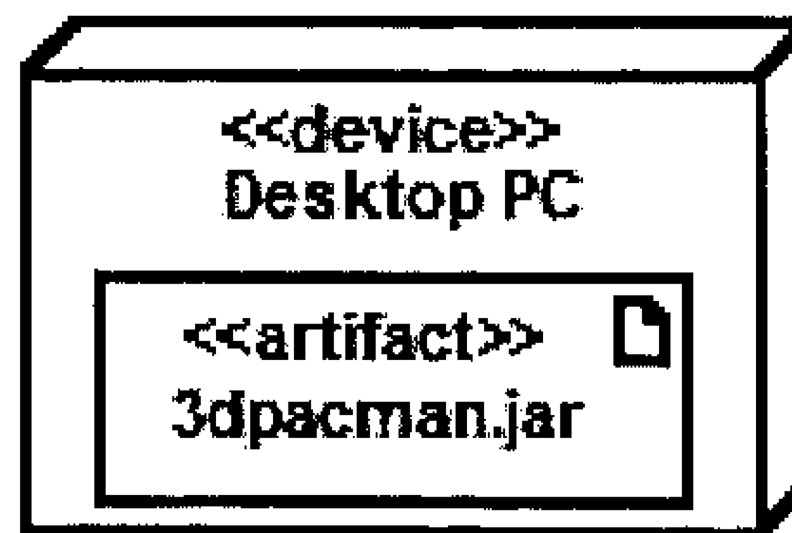
شكل رقم (١٥-٣) نمذجة ملف برمجي مادي (ملف **jar**) بواسطة أداة اصطناعية.

مرة أخرى . . . نمذجة المستويات

يجب رفع مستوى النمذجة مرة أخرى إلى المستوى الصحيح. في الشكل رقم (١٥- ٢)، تم تحديد عقدة جهاز كحاسب شخصي مكتبي. ويعود الأمر كلياً للمنمذج بتحديد مقدار التفاصيل التي يريد إعطاءها لأسماء العُقد. ويمكن أن نكون دقيقين جداً بتحديد الأسماء، مثل الاسم "محطة عمل مع معالج إنتل ٦٤- بت"، كما يمكن أن نكون عامّين جداً، مثل الاسم "حاسب شخصي عام".

إذا كان هناك متطلبات خاصة لأجهزة النظام، فمن المرجح إعطاء أسماء دقيقة جداً للعُقد. وإذا كانت متطلبات الأجهزة غير محددة أو غير هامة، فيمكن حينئذ أن تكون أسماء العقد غامضة. كباقي جوانب لغة النمذجة الموحدة، من المهم التأكّد بأننا ننمذج النظام في المستوى الصحيح.

نحتاج أخيراً إلى وضع هذين الجزأين معاً لإتمام مخطط نشر النظام. ارسم الأداة الاصطناعية داخل العقدة لعرض نشر برنامج اصطناعي في عقدة جهاز. يعرض الشكل رقم (١٥-٤) أن الملف 3dpacman.jar يشتغل على حاسب شخصي مكتبي.



شكل رقم (١٥-٤) يعني رسم أداة اصطناعية داخل العقدة بأنها منشورة فيها.

لكن هل هذا الأمر كامل حقاً؟ ألا نحتاج إلى نمذجة آلة جافا الافتراضية (JVM) لأنه لن تتنفذ شفرة البرنامج من دونها؟ ماذا عن نظام التشغيل؛ أليس هذا مهماً؟ ربما يكون الجواب كذلك للأسف.

يجب أن تحتوي مخططات النشر على تفاصيل مهمة عن النظام بالنسبة للمعنيين بالأمر. وإذا كان من المهم عرض الأجهزة والبرامج المتضمنة و نظام التشغيل وبيئات وقت التشغيل أو حتى برامج سواقات الأجهزة الخاصة بالنظام، فيجب تضمين تلك الأمور في مخطط النشر. كما سيتم عرضه في باقي هذا الفصل، ويمكن استعمال ترميز مخطط النشر لنمذجة كل تلك الأنواع من الأشياء. إذا كان يوجد ميزة غير مهمة بالنسبة للنظام، فمن غير المجدي إضافتها إلى المخطط لأنه سيصبح فوضوياً ويشتت انتباهنا عن الميزّات المهمة في التصميم.

١٥-٢ البرمجيات المنشورة: الأدوات الاصطناعية
Deployed Software: Artifacts

لقد عرض القسم السابق نظرة عامة خفيفة عن الترميز الذي يمكن استعماله لعرض البرامج والأجهزة في النظام المنشور. ولقد تم نشر البرنامج 3dpacman.jar كعقدة جهاز، حيث تتم تسمية الملف JAR المذكور في لغة النمذجة الموحدة بالأداة الاصطناعية artifact.

وتكون الأدوات الاصطناعية ملفات مادية يقوم البرنامج بتنفيذها أو استعمالها. وتتضمن الأدوات الاصطناعية الشائعة التي سنصادفها التالي:

- ملفات قابلة التنفيذ، مثل ملفات exe. أو jar. .
- ملفات المكاتب البرمجية، مثل ملفات dll. (أو ملفات الدعم jar.).
- ملفات برامج المصدر مثل ملفات java. أو cpp..

- ملفات الترتيبات configuration المستعملة من قبل البرامج عند وقت التشغيل، والتي تكون عادة بالصيغ xml.، properties.، أو txt..

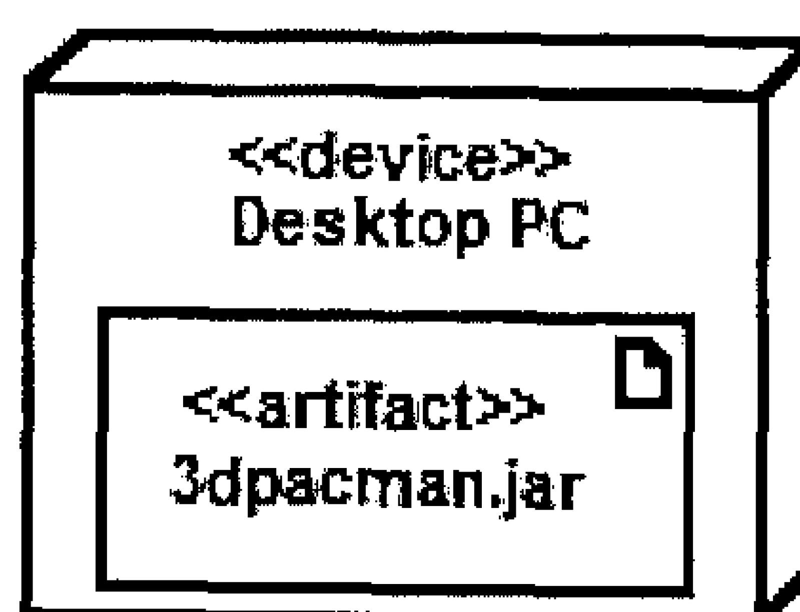
ويتم عرض الأداة الاصطناعية باستعمال مستطيل مع الحاشية <<artifact>>، أو مع أيقونة الوثيقة في الزاوية العليا عن يمين المستطيل، أو مع كليهما، كما هو معروض في الشكل رقم (١٥-٥). في باقي الكتاب، وستعرض الأدوات الاصطناعية باستعمال الحاشية <<artifact>> وأيقونة الوثيقة معاً.



شكل رقم (١٥-٥) التمثيلات المتكافئة للأداة الاصطناعية 3dpacman.jar.

١٥-٢-١ نشر أداة اصطناعية في عقدة
Deploying an Artifact to a Node

يعني نشر الأداة الاصطناعية في العقدة أنها ماكثة أو مُنصّبة فيها. ويعرض الشكل رقم (١٥-٦) الأداة الاصطناعية السابقة 3dpacman.jar المنشور في عقدة حساب شخصي مكتبي، وذلك برسم رمز الأداة الاصطناعية داخل العقدة.



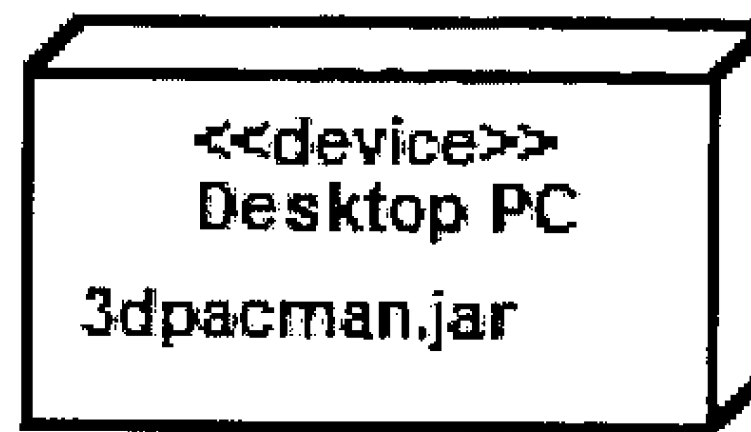
شكل رقم (١٥-٦) نشر الأداة الاصطناعية 3dpacman.jar في عقدة Desktop PC.

يمكنك نمذجة نشر أداة اصطناعية في العقدة بأسلوبَين آخرَيْن. يمكن أيضاً رسم سهم الاعتمادية من الأداة الاصطناعية إلى العقدة الهدف مع الحاشية <<deploy>>، كما هو معروض في الشكل رقم (١٥-٧).



شكل رقم (١٥-٧) طريقة بديلة لنمذجة علاقة النشر.

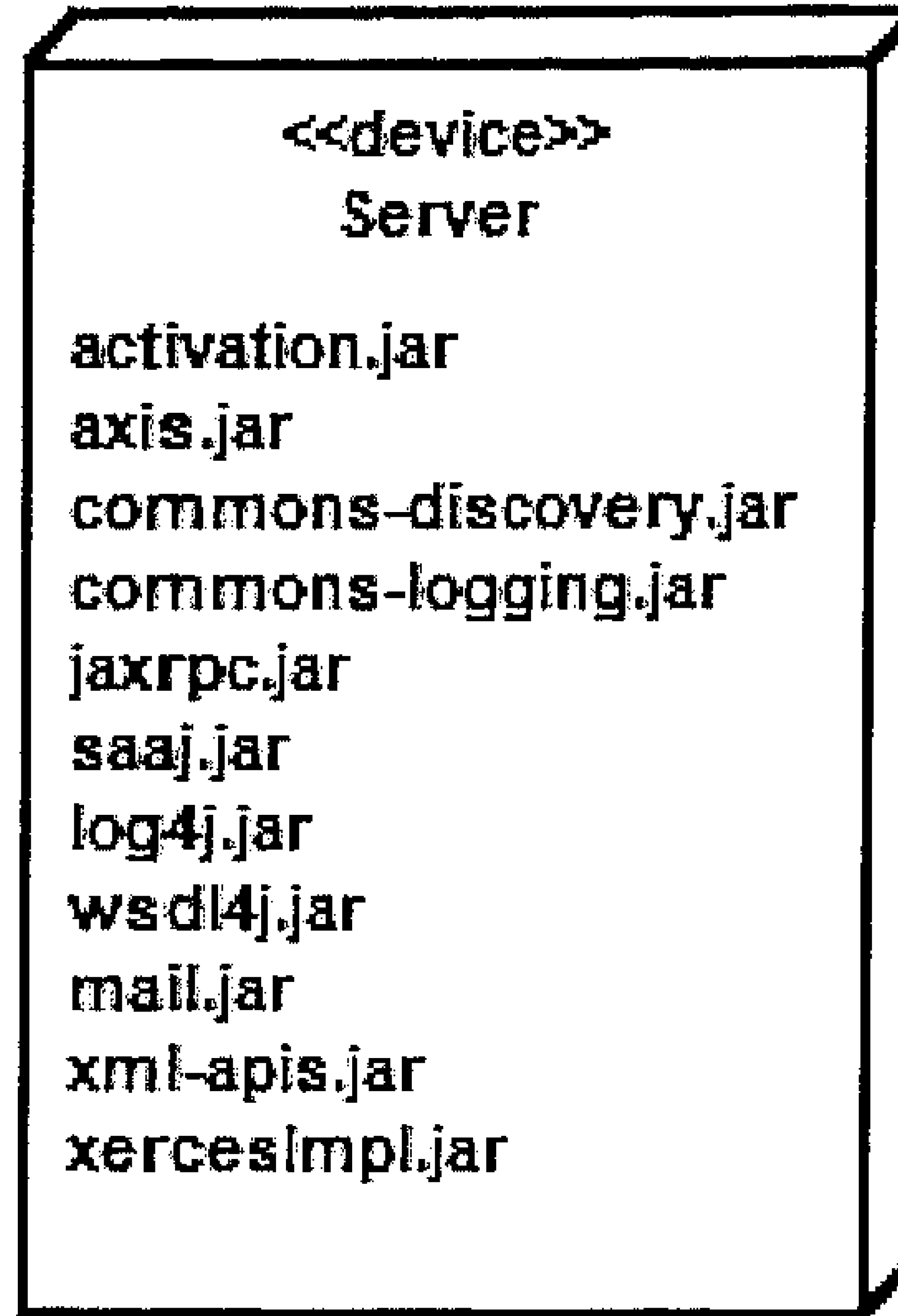
عندما لا يتوفر لدينا مكان رحب في المخطط، فربما أردنا تمثيل النشر بإدراج اسم الأداة الاصطناعية داخل العقدة الهدف، كما هو معروض في الشكل رقم (١٥-٨).



شكل رقم (١٥-٨) طريقة لعرض النشر بإحكام من خلال كتابة اسم الأداة الاصطناعية داخل العقدة.

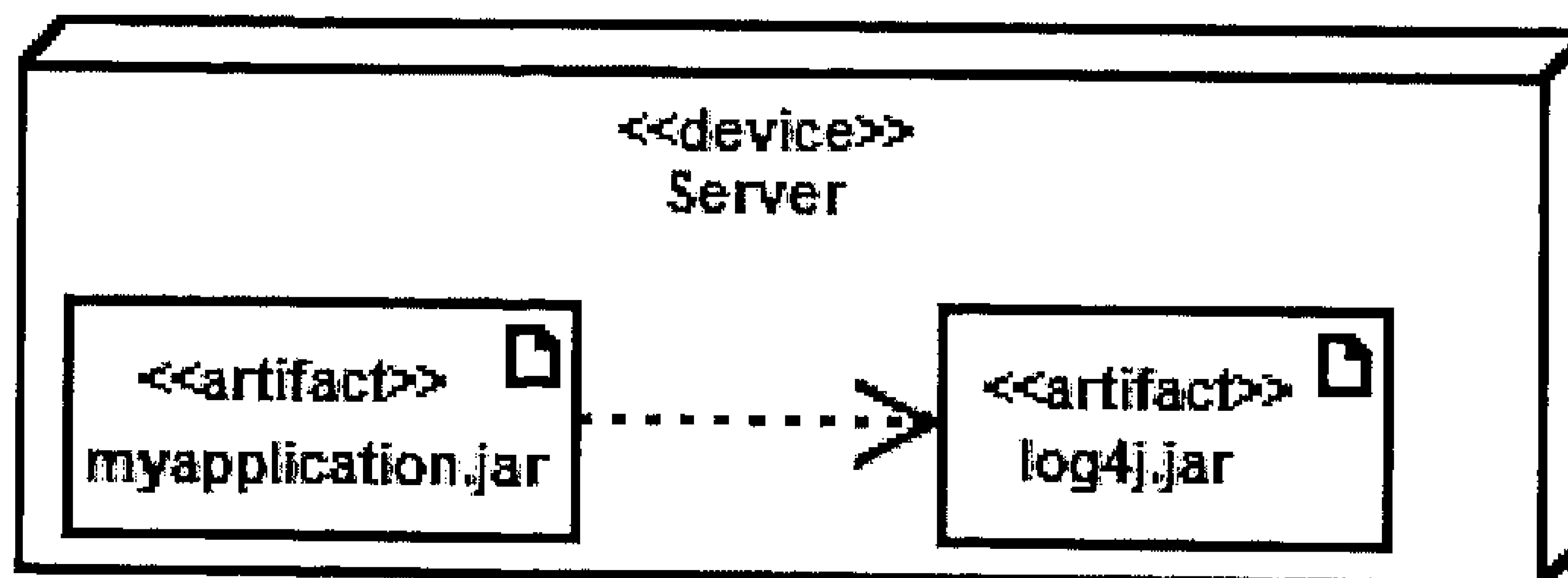
تقوم كل تلك الطرق بعرض نفس علاقة النشر، لذلك نوفر هنا بعض التعليمات لاختيار ترميز ما.

إذا كان عندنا الكثير من الأدوات الاصطناعية، يمكن إدراج الأدوات الاصطناعية (من دون رمز الأداة) لتوفير المكان في المخطط، كما يظهر في الشكل رقم (١٥-٩). تخيّل كم سيصبح المخطط كبيراً إذا قمنا برسم رمز الأداة الاصطناعية لكل من تلك الأداة التي في الشكل رقم (١٥-٩).



شكل رقم (١٥-٩) يوفر إدراج أسماء الأدوات الاصطناعية داخل العقدة الكثير من المكان، وذلك مقارنة برسم رمز الأداة الاصطناعية لكل منها.

ولكن، احذر من أسلوب إدراج الأدوات الاصطناعية؛ لأنه لا يسمح بعرض الاعتماديات التي بين تلك الأدوات. إذا أردت عرض أن أداة اصطناعية تستعمل أداة اصطناعية أخرى، فيجب رسم رموز الأدوات الاصطناعية وسهم الاعتمادية الذي يربط بينهم، كما هو معروض في الشكل رقم (١٥-١٠).



شكل رقم (١٥-١٠) ترميز نشر يستعمل رموز الأدوات الاصطناعية (بدلاً من إدراج أسماء الأدوات) حيث يسمح بعرض الاعتماديات بين تلك الأدوات.

١٥-٢-٢ ربط البرامج بالأدوات الاصطناعية

Tying Software to Artifacts

عند تصميم البرامج، نقوم بتقسيمها إلى مجموعات متماسكة من الناحية الوظيفية، مثل المكوّنات أو الحُزم التي تصبح مُجمّعة بالنهاية في ملف واحد أو أكثر (أو في أدوات اصطناعية). بالتحدث بلغة النمذجة الموحدة، وإذا كانت الأداة الاصطناعية عبارة عن تحقيق مادي لمكوّن ما، فستُظهر الأداة الاصطناعية ذلك المكوّن. ويمكن أن تُظهر الأداة الاصطناعية أيّ عنصر يمكن تحزيمه، مثل الحُزم والأصناف وليس المكوّنات فقط.

وتعرض علاقة الإظهار manifest باستعمال سهم الاعتمادية انطلاقاً من الأداة الاصطناعية إلى المكوّن ومرفقاً بالحاشية <<manifest>>، كما هو معروض في الشكل رقم (١٥-١١).



شكل رقم (١٥-١١) تقوم الأداة الاصطناعية mycomponent.jar بإظهار المكوّن MyComponent.

بما أنه يمكن تعيين أدوات اصطناعية للعُقد، توفر علاقة الإظهار الرابط المفقود في نمذجة كيفية إسقاط مكوّنات البرامج على الأجهزة. وعلى أية حال، قد يؤدي ربط مكوّن ما بأداة اصطناعية وبعقدة إلى الحصول على مخطط غير منظّم، من الشائع إذا عرض علاقات الإظهار منفصلة عن علاقات النشر، حتى إن كانت في نفس مخطط النشر.



يمكن عرض علاقة الإظهار أيضاً في مخططات المكوّنات، وذلك بإدراج الأدوات الاصطناعية التي تظهر المكوّن داخل رمز المكوّن، كما تم مناقشته في الفصل الثاني عشر.

إذا كنت معتاداً على الإصدارات السابقة للغة UML، فربما تكون جربت نمذجة مكوّن يشتغل على جهاز من خلال رسم رمز المكوّن داخل العقدة. ابتداء من UML 2.0، ولقد قامت الأدوات الاصطناعية بدفع المكوّنات نحو تفسير أكثر مفاهيمي، وحيث تُمثل الأدوات الاصطناعية الآن الملفات المادية.

على أية حال، إن كثير من أدوات UML ليست محدّثة بالكامل مع معايير UML 2.0، لذلك يمكن القول إنها ما زالت تستعمل الترميز السابق.

١٥-٣ ما هي العقدة؟

What is a Node?

لقد رأينا سابقاً إمكانية استعمال العُقد لعرض الأجهزة في مخطط النشر، لكن ليس على العُقد أن تكون دائماً عبارة عن أجهزة. يمكن أن تكون بعض أنواع البرامج عُقداً أيضاً (مثل البرامج التي توفر بيئة يتم بداخلها تنفيذ مكونات برامج أخرى).

تكون العقدة عبارة عن موارد أجهزة أو برامج، والتي يمكنها استضافة برامج أو ملفات ذات علاقة. يمكن التفكير بعقدة برنامج كأنها سياق التطبيق؛ لا تكون عادة جزءاً من البرنامج الذي نقوم بتطويره، لكن تكون بيئة طرف ثالث توفر خدمات للبرامج.

تشكل العناصر التالية أمثلة شائعة و معقولة عن عُقد الأجهزة:

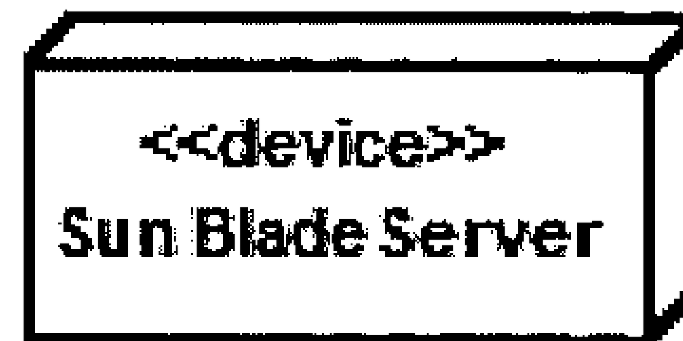
- خادم server
- حاسب شخصي مكتبي Desktop PC
- مشغلات أقراص Disk drives

تشكل العناصر التالية أمثلة عن عُقد بيئة تنفيذ:

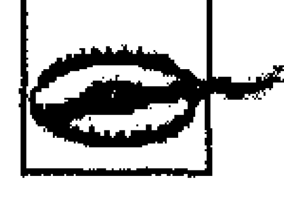
- نظام تشغيل Operating system
- حاوي J2EE (Java 2 Enterprise Edition container)
- خادم ويب Web server
- خادم تطبيق Application server

١٥-٤ عُقد الأجهزة و بيئة التنفيذ
Hardware and Execution Environment Nodes

يتم رسم العقدة على شكل مكعب حيث يتم كتابة نوعها بداخله، كما هو معروض في الشكل رقم (١٥-١٢). تؤكد الحاشية <<device>> على أن العقدة هي عقدة أجهزة.

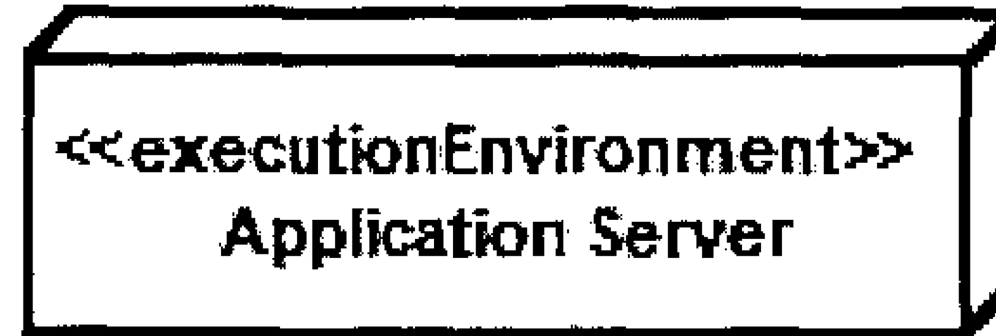


شكل رقم (١٥-١٢) تستخدم الحاشية **<<device>>** لتحديد العقدة **Sun Blade Server**.



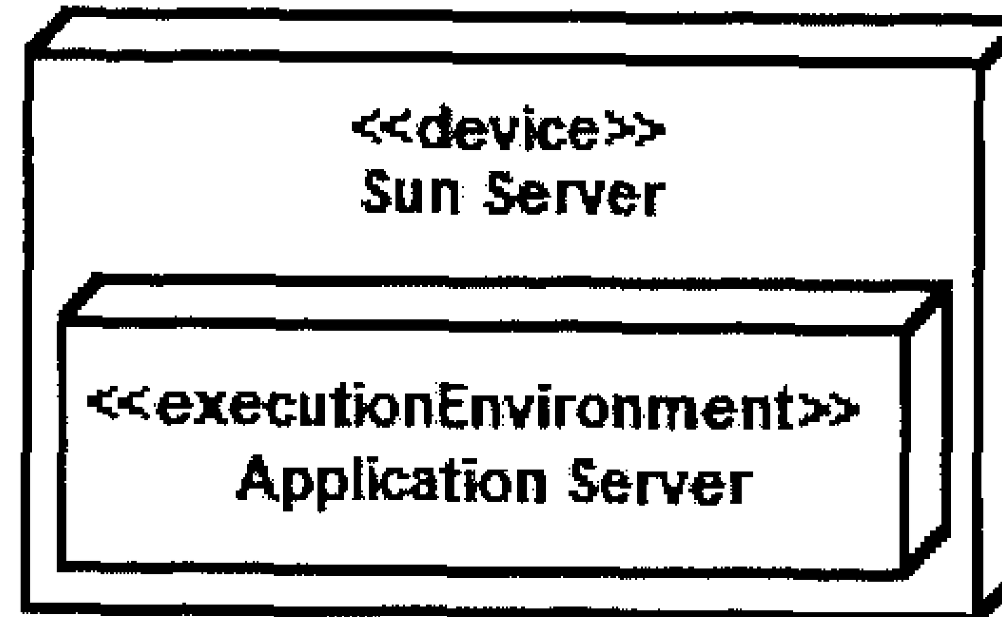
إن عناصـر البرمجيـات كملفـات المكاتـب البرمجيـة و ملفـات الخصـائص والملفات التنفيذية، والتي لا تستطيع استضافة البرمجيات، هـي ليسـت عُقـد بل أدوات اصطناعية (انظر القسم "البرامج المنشورة: الأدوات الاصطناعية" سابقاً في هذا الفصل).

يعـرض الشـكل رقـم (١٥-١٣) عقـدة خـادم تطبيقـات. سـيدرك المطلعون على تطوير المشاريع البرمجية بأن هـذا الأمـر عبارة عن نـوع مـن البيئات التنفيذية لأنه بيئة برنامج يوفر خدمات لتطبيقاتنا. تُؤكد الحاشية <<executionEnvironment>> على أنّ هذه العقدة هي عقدة بيئة تنفيذ.



شكل رقم (١٥-١٣) يتم تحديد العقدة **Application Server** باستعمال الحاشية **<<executionEnvironment>>**.

لا تتواجد بيئات تنفيذ بحد ذاتها (فهي تشتغل على الأجهزة). على سبيل المثال، يحتاج نظام التشغيل إلى جهاز حاسب ليشتغل عليه. ويتم عرض تنصيب بيئة التنفيذ على جهاز محدد بوضع العُقد داخل بعضها، كما هو معروض في الشكل رقم (١٥-١٤).



شكل رقم (١٥-١٤) لقد تم عرض العقدة **Application Server** داخل العقدة **Sun Server**، هذا يعني أن خادم التطبيق **Application Server** يُنفذ على أجهزة خادم **Sun Server**.

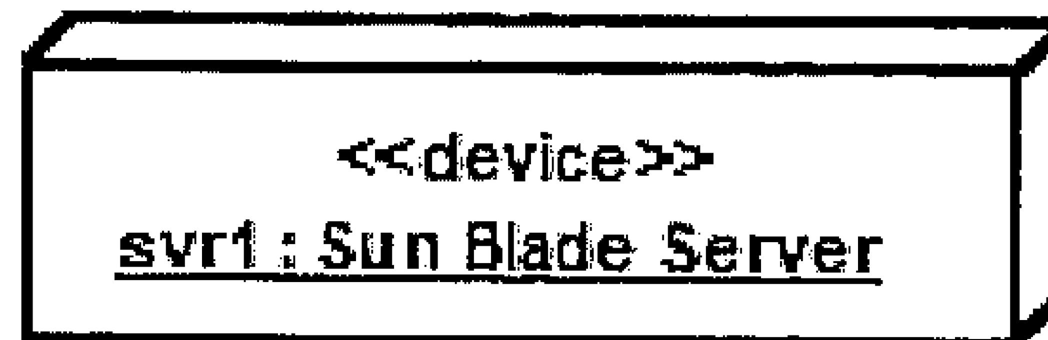
وفي UML 2.0، ليس من الضروري جداً تمييز عُقدة الجهاز عن عُقدة بيئة التنفيذ، ولكن يشكل هذا الأمر ممارسة مفيدة بسبب توضيحه للنموذج.



هل تريد المزيد من التنوع؟ إذا كنت تستعمل المَظهر profile (تمت مناقشته في الملحق B)، يمكن تطبيق حاشيات العقدة التي تكون ذات أهمية أكبر للمجال الخاص بك، مثل الحاشية <<J2EE Container>>. ويمكن تحديد هذه الأنواع الجديدة للعقد في المَظهر الخاص بك كنوع خاص من بيئة التنفيذ.

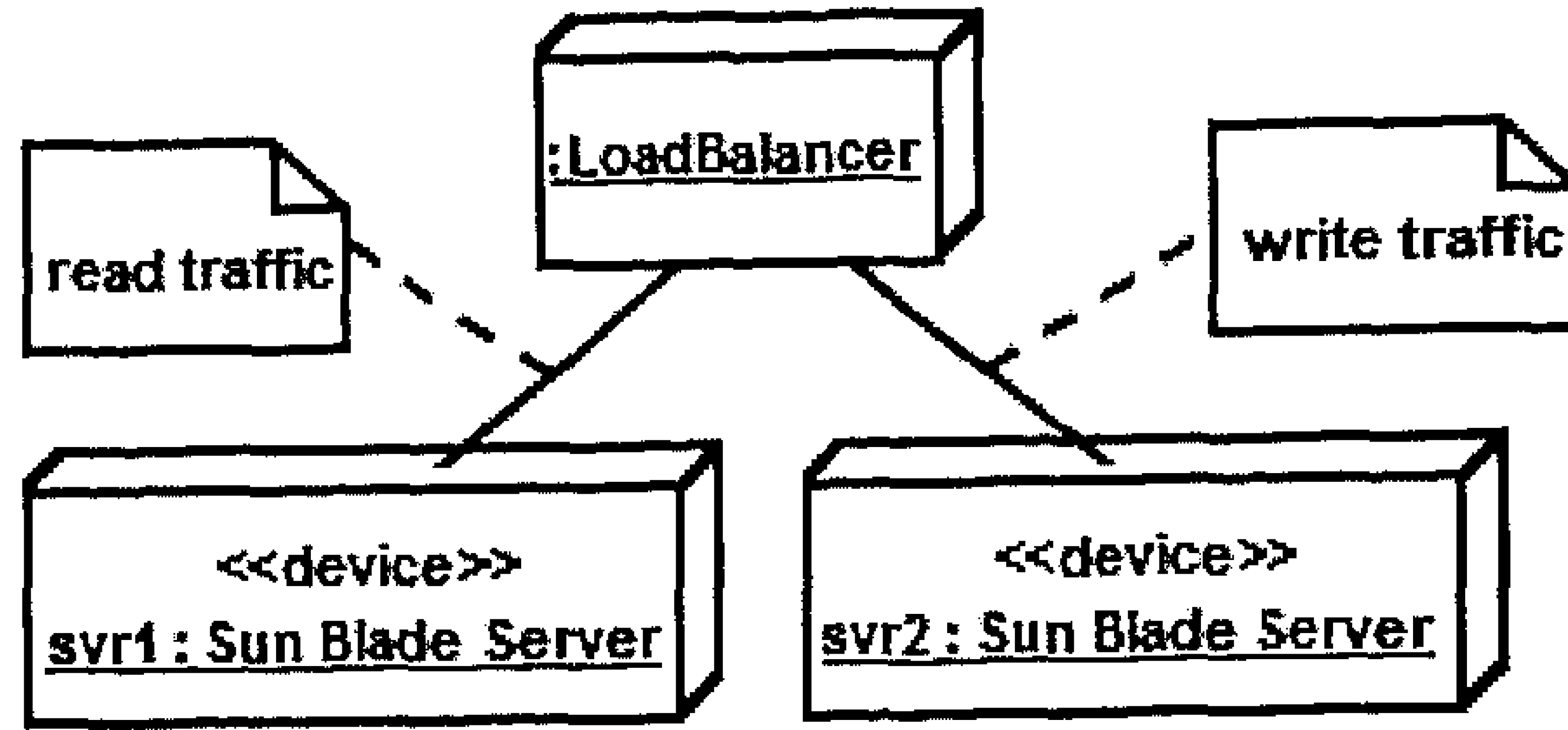
١٥-٤-١ عرض مثيلات العقدة Showing Node Instances

يمكن أن يتكرر تواجد عدة عُقد من نفس النوع في المخطط، لكن نريد لفت الانتباه إلى حقيقة أنها مثيلات مختلفة فعلياً. ويمكن عرض مثيل للعقدة باستعمال الترميز name : type كما في الشكل رقم (١٥-١٥).



شكل رقم (١٥-١٥) عرض اسم و نوع عقدة ما؛ svr1 هو مثيل للخادم Sun Blade.

يعرض الشكل رقم (١٥-١٦) كيفية نمذجة عقدتَين من نفس النوع. لقد تم تعيين أنواع حركة تحميل مختلفة للعُقدتين svr1 و svr2 من خلال ميزان التحميل load balancer (إنها حالة شائعة مع أنظمة الشركات).

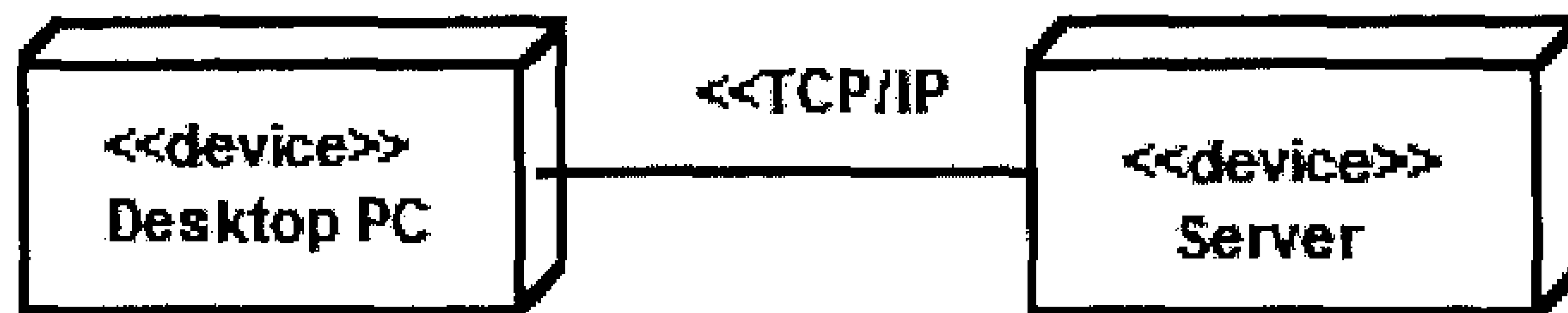


شكل رقم (١٥-١٦) تأخذ عقدة قراءة حركة التحميل read traffic وتأخذ الأخرى كتابة حركة التحميل write traffic.

١٥-٥ الاتصالات بين العُقد
Communication between Nodes

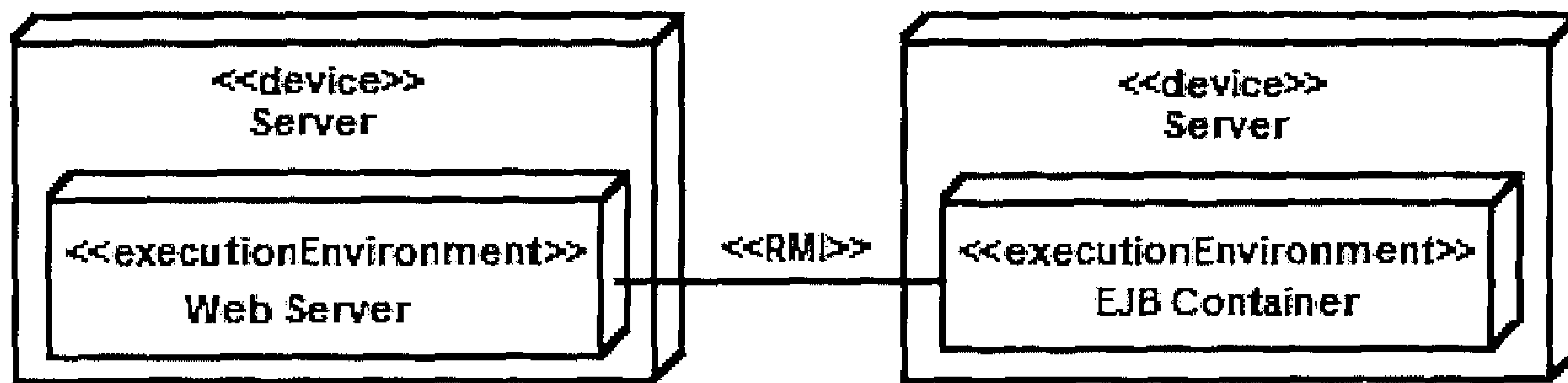
حتى تقوم العُقد بعملها بشكل جيد، ربما تحتاج العقدة الاتصال بعُقد أخرى. على سبيل المثال، قد يقوم تطبيق العميل الذي يشتغل على حاسب شخصي مكتبي باسترجاع البيانات من خادم ما باستعمال بروتوكول الاتصالات TCP/IP.

ويتم استعمال مسارات الاتصالات لعرض تلك العُقد وهي تتصل بعضها ببعض عند وقت التشغيل. ويتم رسم مسارات الاتصالات كخط غير منقط يربط بين عقدتَين. يتم عرض نوع الاتصالات بإضافة حاشية إلى المسار. ويعرض الشكل رقم (١٥-١٧) عقدة حاسب شخصي مكتبي وعقدة خادم (تتصلان فيما بينهما باستعمال TCP/IP).



شكل رقم (١٥-١٧) حاسب شخصي مكتبي وخادم يتصلان عن طريق TCP/IP.

يمكن أيضاً عرض مسارات الاتصالات بين عُقد بيئة التنفيذ. على سبيل المثال، يمكن نمذجة خادم ويب يتصل بحاوية EJB من خلال أسلوب الاستدعاء عن بعد للآلة Remote Machine Invocation (RMI)، كما هو معروض في الشكل رقم (١٥-١٨). ويكون هذا أكثر دقة من عرض مسار اتصالات RMI على مستوى عقدة الجهاز، وذلك بسبب "تخاطب" عُقد بيئة التنفيذ بأسلوب RMI. وعلى أية حال، يرسم بعض المنمذجين مسارات الاتصالات على مستوى العقدة الخارجية لأنه يجعل المخطط أقل فوضوية.



شكل رقم (١٥-١٨) يمكن أيضاً عرض مسارات الاتصالات بين عُقد بيئة التنفيذ.

من الصعب أحياناً تعيين حاشية لمسار الاتصالات. ويكون الاستدعاء RMI ذو طبقات باستعمال طبقة النقل TCP/IP. لذلك، هل يجب علينا تخصيص حاشية <<RMI>> أم حاشية <<TCP/IP>>؟ حسب الخبرة، يجب أن تكون حاشيات الاتصالات الخاصة بنا عالية المستوى قدر الإمكان لأنها تتواصل بدرجة كبيرة بخصوص النظام. في هذه الحالة، تكون الحاشية <<RMI>> هي الاختيار الصحيح؛ فهو مستوى عالٍ ويُخبر القارئ بأننا نستعمل تطبيق جافا. على أية حال، يجب تكييف المخطط لملاءمة جمهورنا، كما هو حال النمذجة مع UML.



تبين مسارات الاتصالات قدرة العُقد على الاتصال بعضها ببعض و هي ليست معدّة لعرض الرسائل الفردية، مثل الرسائل في مخطط التتابع.

ابتداء من UML 2.0، من المفترض تحديد الحاشيات في مظهر ما profile، لذلك من الناحية النظرية، يجب استعمال الحاشيات التي يوفرها المظهر الخاص بنا فقط. وعلى أية حال، حتى إذا كنا لا نستعمل مظهراً محدداً، فقد تسمح لك أداة UML المستعملة بإنشاء أي حاشية. وبما أن الحاشيات هي وسيلة جيدة لعرض أنواع الاتصالات في النظام، فاعمل على سجيتك عند الضرورة وإذا كانت الأداة المستعملة تسمح بذلك. لكن إن قمت بذلك، فحاول أن تبقي الحاشيات متماسكة. على سبيل المثال، لا تقوم بإنشاء الحاشية <<Remote Method Invocation>> والحاشية <<RMI>> لأنهما تمثلان نفس نوع الاتصالات.

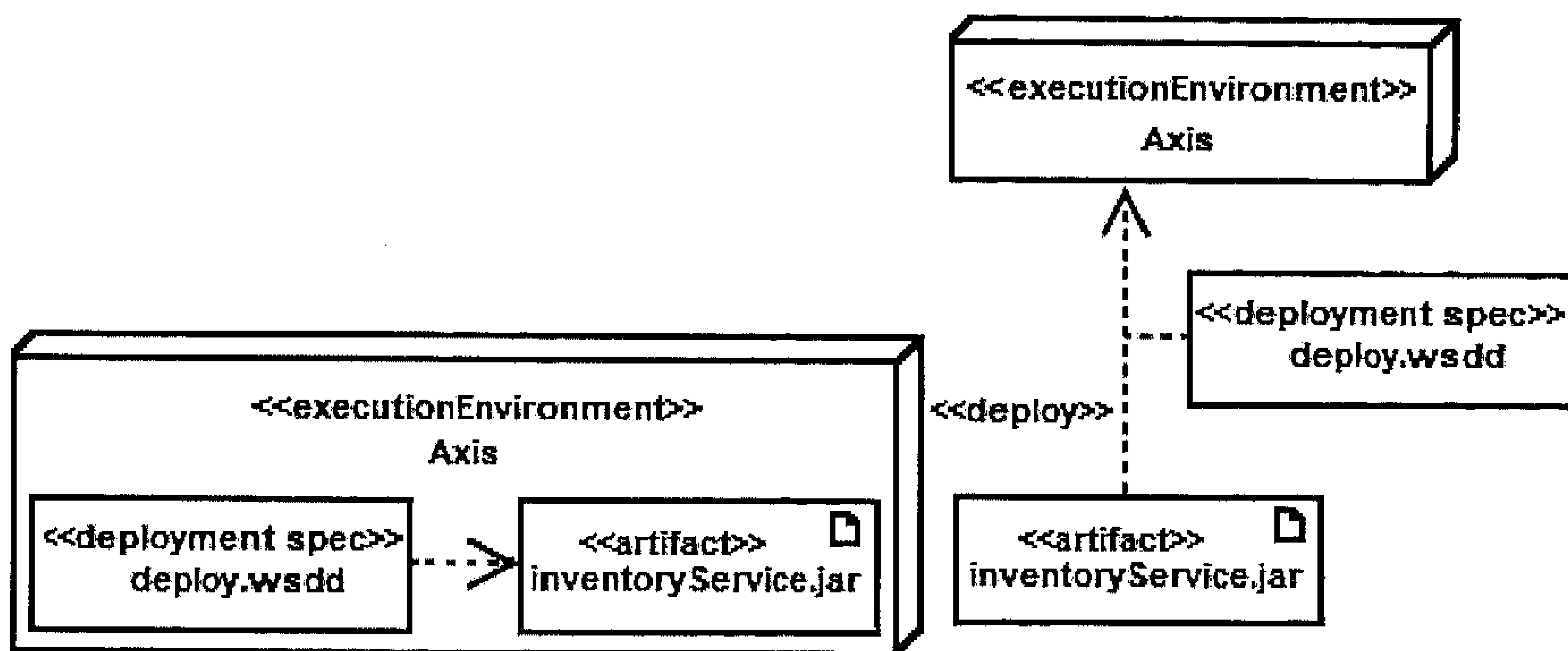
١٥-٦ مواصفات النشر
Deployment Specifications

من النادر أن يكون تنصيب البرامج سهلاً كعملية إسقاطه ملف ما على الجهاز؛ عادة ما يكون علينا تحديد بارامترات الإعدادات قبل أن يصبح بالإمكان تنفيذ البرامج. إن مواصفات النشر هي: أداة اصطناعية خاصة تقوم بتحديد كيفية نشر أداة اصطناعية أخرى في عقدة ما. وهي توفر معلومات تسمح للأداة الاصطناعية الأخرى بأن تشتغل بنجاح في بيئتها.

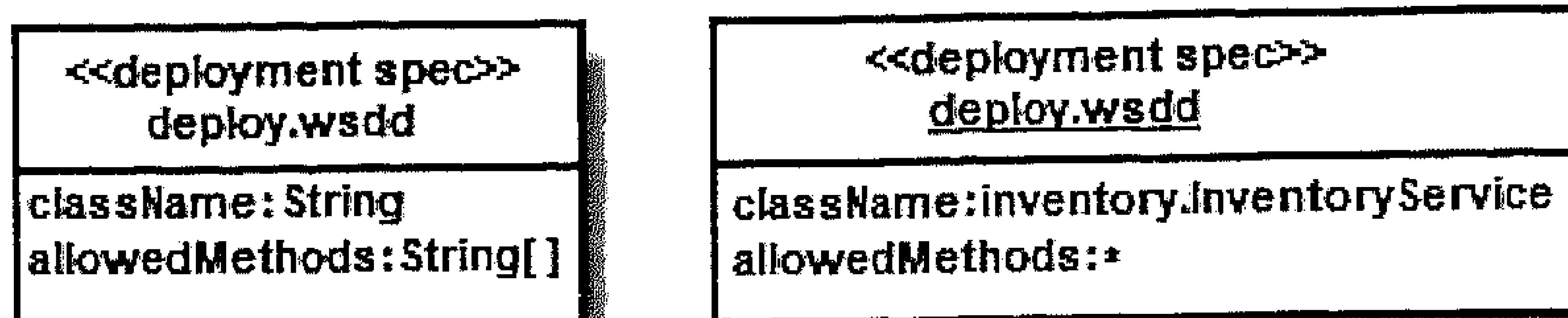
يتم رسم مواصفات النشر على شكل مستطيل مع الحاشية <<deployment spec>>. وهناك طريقتان لربط مواصفات النشر بالنشر التي تصفه:

- رسم سهم اعتمادية من مواصفات النشر إلى الأداة الاصطناعية، مع وضعهما داخل عقدة الهدف.
- ربط مواصفات النشر بسهم النشر، كما هو معروض في الشكل (١٥-١٩).

إن الملف deploy.wsdd (المعروض في الشكل رقم ١٥-١٩) هو ملف وصف النشر القياسي الذي يحدد كيفية نشر خدمة وِبْ على محرّك خدمة وِبْ اكسس Axis. ويوضح هذا الملف أي صنف ينفذ خدمة الوِبْ وأي طُرق في الصنف يمكن استدعاؤها. ويمكن إدراج هذه الخصائص في مواصفات النشر باستعمال الترميز name : type. يعرض الشكل رقم (١٥-٢٠) مواصفات النشر deploy.wsdd باستعمال الخصائص allowedMethods وclassName .



شكل رقم (١٥-١٩) أساليب متساوية في ربط مواصفات النشر بالنشر التي تصفه.



شكل رقم (١٥-٢٠) خصائص مواصفات النشر مع قيم مثيل لها في الترميز عن اليمين.

يعرض الترميز عن اليمين مثيل مع قيمه لمواصفات النشر. استعمل هذا الترميز إذا أردت عرض القيم الحالية للخصائص بدلاً من الأنواع فقط. لست بحاجة إلى إدراج كل خاصية في مواصفات الانتشار، إنما فقط الخصائص التي تعتبرها مهمة للانتشار. على سبيل المثال، وقد يحتوي الملف deploy.wsdd على خصائص أخرى مثل الأدوار المسموح بها، ولكن إذا كنت لا تستعمل تلك الخاصية أو هي غير مهمة (إنها نفسها لكل خدمات الويب خاصتك) فاتركها حينئذ.



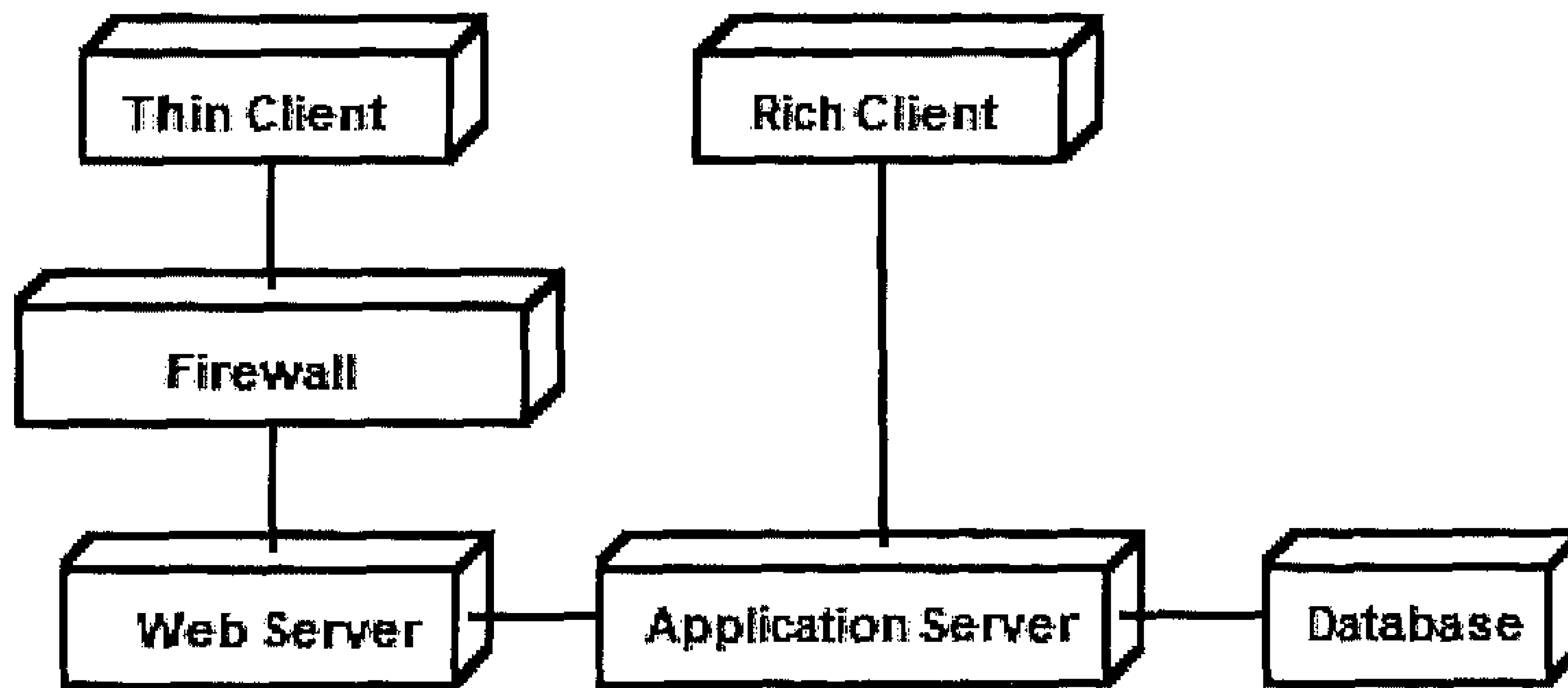
لقد قمنا في هذا الفصل بالإشارة بإيجاز إلى مثيلات العناصر في مخططات النشر، لكن يمكن نمذجة مثيلات العُقد والأدوات الاصطناعية و مواصفات النشر. في مخططات الانتشار، لا يهتم العديد من المنمذجين بتحديد أن العنصر هو مثيل عندما يكون المعنى واضحاً. على أية حال، إذا أردت تحديد قيم خصائص مواصفات النشر (مثل الجانب الأيمن من الشكل رقم ١٥- ٢٠)، فتكون هذه حالة نادرة حيث قد تُجبرك أداة UML على استعمال ترميز المثيل. لا تدعم حالياً معظم أدوات UML رمز مواصفات النشر. يمكن تعويض ذلك بالربط بملاحظة تحتوي على معلومات مماثلة.

١٥-٧ متى نستعمل مخطط النشر؟
When to Use a Deployment Diagram?

تفيد مخططات النشر في كل مراحل عملية التصميم. عندما تبدأ بتصميم نظام ما، ربما تكون تعرف المعلومات الأساسية فقط عن الترتيبات المادية. على سبيل المثال، إذا كنت تقوم بإنشاء تطبيق وِبْ، ربما لم تقرر بعد بخصوص الأجهزة التي ستستعملها وعلى الأرجح لا تعرف أسماء أدواتك البرمجية الاصطناعية. لكن تريد التحدث عن خصائص النظام المهمة، مثل التالي:

- اشتمال المعمارية على خادم وِبْ وخادم تطبيق وقاعدة بيانات.
- إمكانية وصول العملاء إلى التطبيقات من خلال متصفح وِبْ، أو من خلال واجهة رسومية أكثر غنى.
- حماية خادم الوِبْ بواسطة جدار ناري.

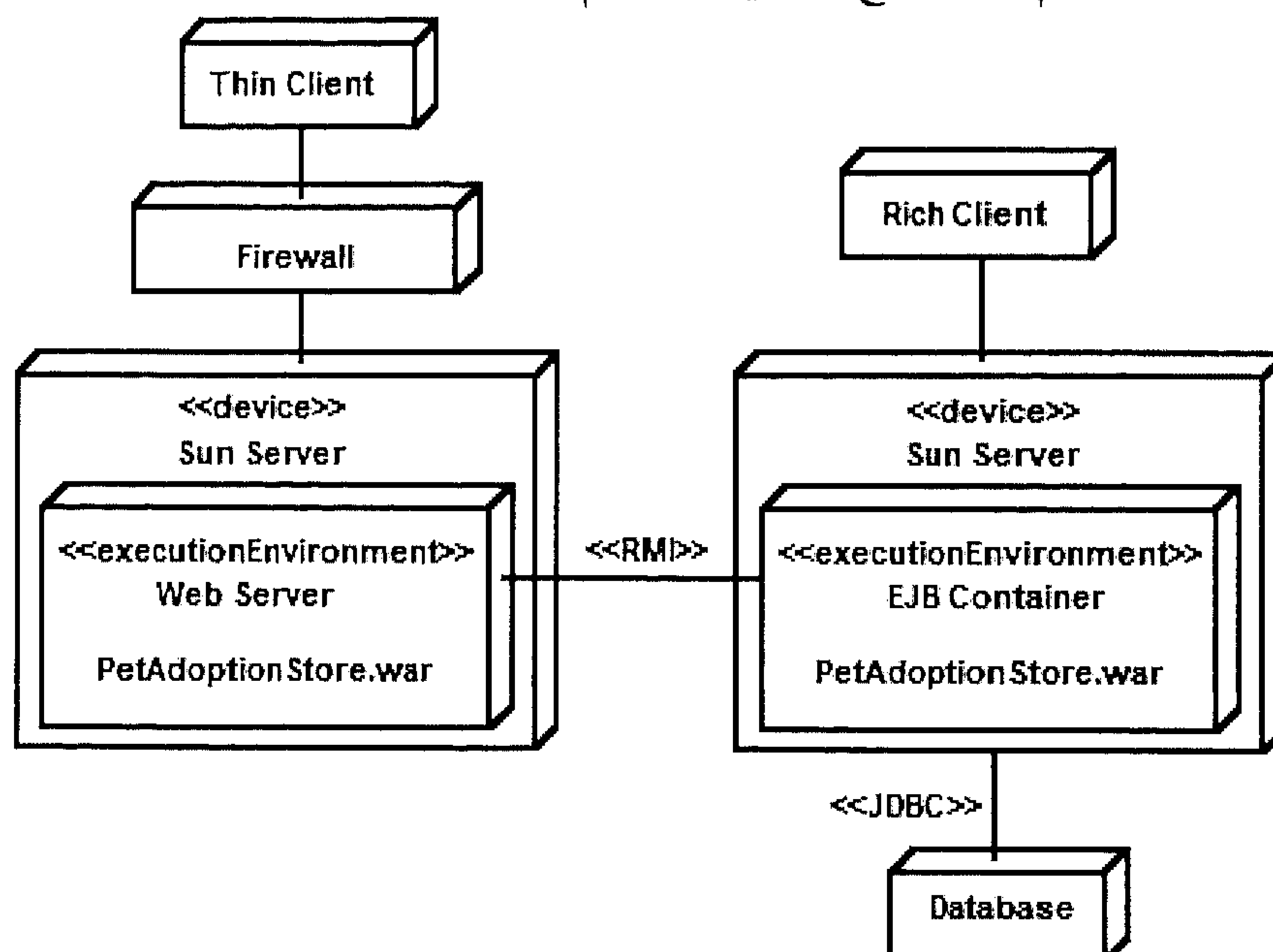
حتى في هذه المرحلة المبكرة، يمكن استعمال مخططات النشر لنمذجة هذه الميزات. يعرض الشكل رقم (١٥- ٢١) مخطط أولي لأفكار النظام الرئيسية. ليس علينا تحديد أسماء العُقد ولا تحديد بروتوكولات الاتصالات.



شكل رقم (١٥-٢١) مخطط أولي للأفكار الرئيسية لتطبيق ويب خاص بك.

تفيد مخططات النشر في المراحل اللاحقة من تطوير البرامج. ويعرض الشكل رقم (١٥-٢٢) مخطط نشر مفصّلاً حيث يحدد إنجاز النظام باستعمال J2EE. ويعتبر أيضاً أكثر تحديداً بالنسبة لأنواع الأجهزة وبروتوكولات الاتصالات وتوزيع الأدوات البرمجية الاصطناعية على العُقد. ويمكن استعمال هكذا مخطط كطبعة كربونية عن كيفية تنصيب النظام.

يمكن العودة من جديد إلى مخططات النشر أثناء تصميم النظام لصقل المخططات الأولية العامة، وذلك بإضافة التفصيلات، كما سنختار التقنيات وبروتوكولات الاتصالات والأدوات البرمجية الاصطناعية التي سنستعملها. وتسمح مخططات النشر المصقولة بالتعبير عن الرؤية الحالية لترتيبات النظام المادية مع المعنيين بالنظام.



شكل رقم (١٥-٢٢) يمكن توفير أي قدر من التفاصيل عن التصميم المادي للنظام.

١٥-٨ ما هي الخطوة التالية؟

لقد أنهيت تعلم المفاهيم الأساسية للغة النمذجة الموحدة، لكن واصل قراءة الملاحق للحصول على ملخص عن بعض تقنيات النمذجة المتقدّمة. وتقدّم الملاحق لغة قيود الكائن (OCL) التي تشكل وسيلة حاسمة لعرض القيود في المخططات، وتقدم المَظاهر profiles التي تسمح بتعريف واستعمال مفردات متعارف عليها للغة النمذجة الموحدة. ومن المفيد مراجعة تلك الملاحق للحصول على شعور بدقة إضافية يمكن إضافتها إلى نموذجك، والحصول على قدرات إضافية ناتجة عن تلك الدقة. وستتم تغطية لغة قيود الكائن في الملحق (أ)، وسيتم وصف المظاهر في لغة النمذجة الموحدة في الملحق ب.

الملاحق

أ- لغة قيود الكائن
Object Constraint Language (OCL)

ب- تكييف لغة النمذجة الموحدة: المظاهر
Adapting UML: Profiles

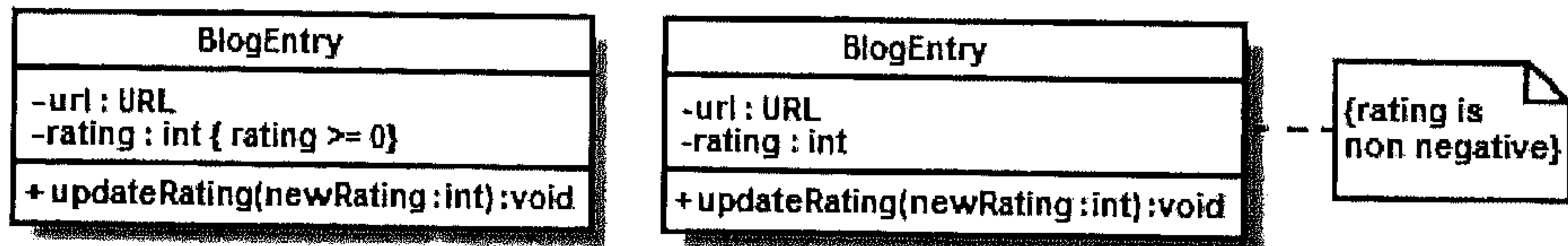
ج- لمحة تاريخية عن لغة النمذجة الموحدة
A History of UML

لغة قيود الكائن

OBJECT CONSTRAINT LANGUAGE (OCL)

قدم الفصل الخامس كتابة القيود على مخططات الأصناف باستعمال لغة قيود الكائن. ولسنا مجبرين على استعمال لغة قيود الكائن للتعبير عن القيود ، حيث يمكننا القيام بذلك باستعمال تركيبات لغة البرمجة المفضلة أو حتى باستعمال اللغة الطبيعية. ويناقش هذا الملحق فوائد وميزات لغة قيود الكائن ويوفر تفاصيل إضافية للإكثار من استعمالها.

تَذَكَّر من الفصل الخامس أنه تتم كتابة القيود داخل الأقواس [ ] بعد العنصر الذي نضع له القيد ، أو يتم إظهارها داخل ملاحظة مرفقة. ويعرض الشكل رقم (أ-١) أساليب مختلفة لتحديد وجوب أن تكون الخاصيّة تقدير rating غير سالبة.



شكل رقم (أ-١) أساليب مختلفة لإرفاق القيود و التعبير عنها.

يبين الشكل رقم (أ-١) أنه يمكن أن تختلف الكلمات المعبرة عن القيد. ويمكن كتابة القيود باللغة الطبيعية، مثل:

التصنيف أكبر من أو يساوي صفر

rating is greater than or equals to zero

كما ويمكن أيضاً كتابة القيود كتعابير لغة برمجة، مثل:

التقدير=< **0** (أي التقدير أكبر من أو يساوي صفر)

rating >= 0

بما أنه يمكن أن تكون اللغة الطبيعية غامضة (وطويلة كثيراً!)، فيستعمل العديد من المنمذجين تركيباً نحوياً مشابهاً للغة البرمجة المفضلة لديهم: لاحظ أن التعبير rating >= 0 يشبه التعبيرات بلغة جافا أو C.

ويمكن أن تصبح القيود أكثر تعقيداً؛ على سبيل المثال، يمكن أن تحدّد أن قيمة معيّنة ليست معدومة null (تستعمل القيمة null مع المؤشرات في البرمجة لتحديد أنها لا تشير إلى قيمة فعلية). وهذا يعني أن لدينا كثيراً من الخيارات للتعبير عن القيود وصياغتها، فكيف نختار الترميز الذي سنستعمله؟ قد تختلف تلك التعابير من لغة برمجة إلى أخرى. إذا تم التعبير عن القيود بطريقة قياسية ومتوقّعة، فيمكننا حينها فهم القيود بسهولة، ويمكن أن تفهما الأدوات الآلية أيضاً. يسمح هذا بإجراء تدقيق آلي للقيود في المخططات و في الشفرة المولدة منها.

بناء على ما سبق، اقتنعت المجموعة OMG (المجموعة المسؤولة عن UML) بوجود حاجة إلى لغة قيود رسمية فريدة، ولكن هناك متطلبات

خاصة لتلك اللغة. ويجب أن تسمح هكذا لغة بالتحقّق من القِيَم و ليس بتغيرها، وعلى اللغة أن تكون لغة تعبير **expression language**. ويجب أن تكون اللغة عامّة بما فيه الكفاية للتمكن من استعمالها في التعبير عن القيود في مخططات UML، وذلك بغض النظر عن لغة البرمجة المستهدفة أو المستعملة. وأخيراً، لا بُدّ من أن تكون اللغة بسيطة كفاية حتى يتمكن الناس من استعمالها حقاً، و هذا ليس صحيحاً بالنسبة للعديد من اللغات الرسمية.

ولقدتم تطوير لغة قيود الكائن في شركة IBM، حيث تمتعت بكل هذه الميزّات مما جعلها ملائمة كلياً. لذلك تم اختيار لغة قيود الكائن للعمل بجانب لغة النمذجة الموحدة من أجل توفير لغة رسمية سهلة الفهم قادرة على تحديد القيود.

وليس من الضروري استعمال لغة قيود الكائن. بشكل عام، يتعلق قرار استعمال المنمذجين للغة قيود الكائن بمجموعة من العوامل، حيث تتضمن مدى توسع نطاق النمذجة الذي توفره ومدى الأهمية التي توليها للتصميم بالتعاقد design by contract (مناقش لاحقا). إذا كانت هذه العوامل تنطبق عليك، فيستحق الأخذ بالاعتبار لغة قيود الكائن لأن التحقق الآلي من القيود يمنح سلامة أكبر للنموذج.

ولقد تم استعمال لغة قيود الكائن بشكل أساسي في مخططات لغة النمذجة الموحدة، وذلك من أجل كتابة القيود في مخططات الأصناف و شروط الحرّاس في مخططات الحالة و النشاط.

أ-١ بناء تعابير لغة قيود الكائن
Building OCL Expressions

يعرض الشكل رقم (أ-٢) مخطط أصناف مع بضعة تعابير لغة قيود الكائن حيث تتضمّن:

- مقارنة عددية بسيطة:

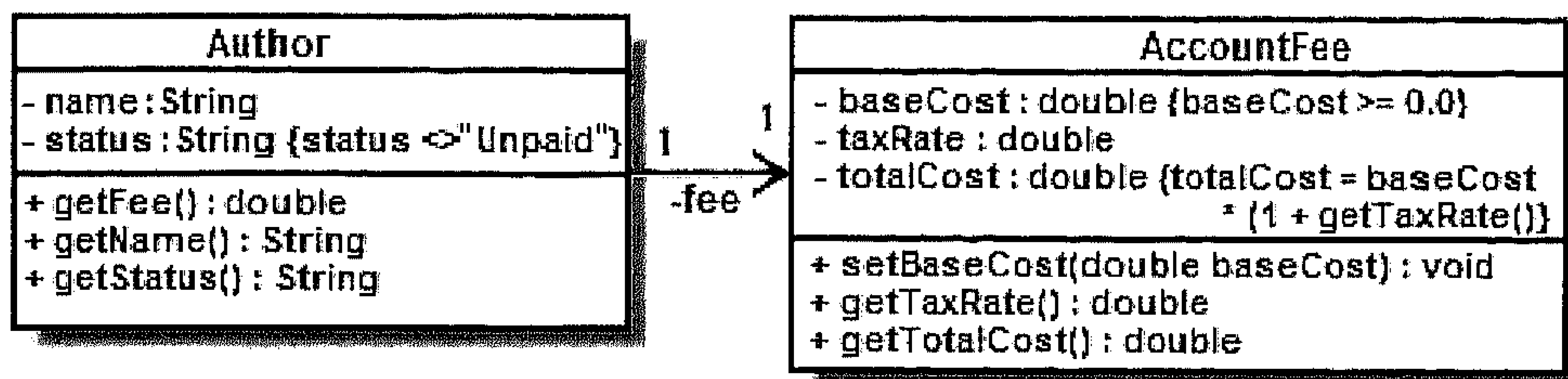
baseCost >= 0.0

- مقارنة عددية أكثر تعقيداً:

totalCost = baseCost * (1 + getTaxRate())

- مقارنة سلاسل رموز:

Status <> "Unpaid"



شكل رقم (أ-٢) مثال عن قيود لغة قيود الكائن بتعقيدات مختلفة.

تتألف تعابير لغة قيود الكائن من عناصر النموذج والثوابت والعوامل. وتتضمّن عناصر النموذج خصائص وعمليات الأصناف، والأعضاء المدرجة من خلال الشراكة. وتستعمل تعابير لغة قيود الكائن التي في الشكل رقم (أ-٢) عناصر النموذج baseCost و totalCost و getTaxRate(). (تحتوي الأقسام التالية على تعابير لغة قيود الكائن مع أعضاء مدرجة من خلال الشراكة).



بخلاف العديد من لغات البرمجة، مثل جافا، يتم استعمال العامل = في لغة قيود الكائن لاختبار المساواة بين عنصرين وليس لإسناد قيمة.

تكون الثوابت عبارة عن قيم ثابتة من أحد أنواع لغة قيود الكائن سابقة التعريف. يتضمّن الشكل رقم (أ-٢) الثابت 0.0 من نوع الأعداد الحقيقية Real، و يتضمن الثابت "Unpaid" من نوع سلاسل الرموز String. وتجمع العوامل بين عناصر النموذج و الثوابت لتشكل التعابير. يضم الشكل رقم (أ-٢) العوامل <>، + و =.

تناقش الأقسام التالية أساسيات أنواع بيانات لغة قيود الكائن و عوامل، ثم تعرض كيفية دمج تلك العناصر داخل التعابير التي يمكن استعمالها في نماذج لغة النمذجة الموحدة.

أ-٢ أنواع البيانات Types

تضم لغة قيود الكائن أربعة أنواع بيانات ضمنية: Boolean (منطقي) و Integer (عدد صحيح) و Real (عدد حقيقي) و String (سلسلة رموز). ويعرض الجدول رقم (أ- ١) بعض الأمثلة عن قيم نموذجية لهذه الأنواع يمكن مصادفتها في تعابير لغة قيود الكائن.

جدول رقم (أ-١) أنواع البيانات المتضمنة في لغة قيود الكائن.

النوع	أمثلة عن قيمها
Boolean	true ; false
Integer	1; 6664; -200
Real	2.7181828; 10.5
String	"Hello, World."

أ-٣ العوامل Operators

تضم لغة قيود الكائن العوامل الأساسية للعمليات الحسابية والعمليات المنطقية، وعمليات المقارنة وتضم أيضاً دوال متقدمة، مثل إرجاع القيمة الكبرى من بين قيمتين ووصل سلاسل الرموز concatenate. وتراقب لغة قيود الكائن أنواع البيانات التي تتطبق عليها العوامل (typed language) لذلك يجب أن يكون هناك معنى لتطبيق العوامل على بيانات محددة. على سبيل المثال، لا نستطيع أن نجمع عدداً صحيحاً وقيمة منطقية. يعرض الجدول رقم (أ-٢) العوامل شائعة الاستعمال في تعابير لغة قيود الكائن.

جدول رقم (أ-٢) العوامل شائعة الاستعمال في تعابير لغة قيود الكائن.

المجموعة	العوامل	تستعمل مع الأنواع	التعبير
حسابية	+, -, *, /	Integer, Real	baseCost + tax
حسابية جمعية	abs(), max(), min()	Integer, Real	score1.max(score2)
مقارنة	<, <=, >, >=	Integer, Real	rate > .75
مساواة	=, <>	All types	age = 65 title <> "CEO"
منطقية	and, or, xor, not	Boolean	isMale and (age >= 65)
سلاسل رموز	concate(), size(), substring(), toInteger(), toReal()	String	Title.substring(1,3)

ترجع عوامل المقارنة و المساواة والمنطق قيمة منطقية (صح أو خطأ). على سبيل المثال، تكون قيمة التعبير age = 65 صح أو خطأ. وترجع العوامل الأخرى في الجدول رقم (أ-٢) قيمة من نفس نوع البيانات المطبّقة عليها. على سبيل المثال، إذا كان baseCost و tax من النوع عدد حقيقي، فتكون نتيجة baseCost + tax من النوع عدد حقيقي أيضاً.

يبين الشكل رقم (أ-٢) أن ()getTaxRate ترجع قيمة double (أي عدد حقيقي مضاعف الدقة، تم كتابة هذا النموذج باستعمال أنواع بيانات لغة جافا)، لكن يبين الجدول رقم (أ-٢) أن العامل + مُعرّف على الأعداد الحقيقية وعلى الأعداد الصحيحة. هذا جيد تماماً؛ عن بناء تعابير لغة قيود الكائن، يمكن مطابقة أنواع البيانات مع أنواع بيانات لغة قيود الكائن الأقرب إليها.



تسمح لغة قيود الكائن أيضاً بالتعبير عن عوامل المجموعات collections مثل الاتحاد unions of sets. ويمكن الرجوع إلى الكتاب UML 2.0 in a Nutshell (O'Reilly) للحصول على قائمة شاملة عن تعابير لغة قيود الكائن.

أ-٤ دمجها معاً Pulling It Together

لقد رأيت حتى الآن وحدات بناء تعابير لغة قيود الكائن. دعنا الآن ندمجها معاً لبناء مثال عن تلك التاعبير.

totalCost = baseCost * (1 +getTaxRate())

لقد أخذنا هذا التعبير من الشكل رقم (أ-٢). إنه يحتوي على وحدات بناء تعابير لغة قيود الكائن التالية:

عناصر نموذجية: **totalCost** ، **baseCost**و **getTaxRate()**

الثوابت: **1**

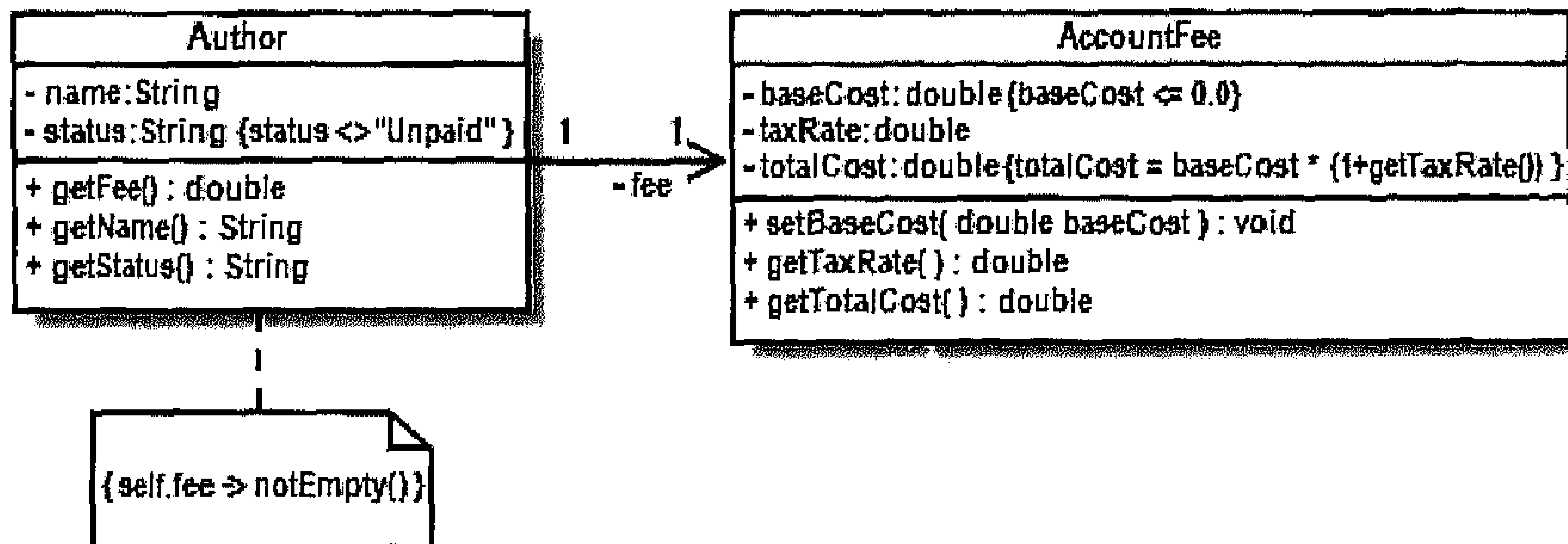
العوامل: = ، * و +

يتألف التعبير أعلاه من عدّة تعابير لغة قيود الكائن قد تم دمجها تباعاً بواسطة العوامل. على سبيل المثال، يتم تقييم **1 + getTaxRate()** إلى

عـدد حقيقـي Real، ثـم يـتم ضـرب النتيجـة بــ **baseCost**. ويـتم بعـد ذلـك فحص القيمة الناتجة عن الضـرب إذا كانت تساوي قيمـة **totalCost** أم لا من خلال استعمال عامل المساواة =. يمكن دمج عناصر النموذج و الثوابت و التعابير وفقاً لأنواعها، ولكن يجب على التعابير المُدمجة أن تكـون مـن النوع منطقي. هذا لأننا نُركّز على استعمال لغة قيود الكائن للتعبير عن القيـود و الحرّاس الـتي يجـب أن تكـون قيمهـا مـن النـوع منطقـي (صـح أو خطأ).

هنـاك قيـد آخـر يسـتعمل بشـكل شـائع لتحديـد أن قيمـة كـائن ليست null. من أجل تحديد ذلك علينـا استعمال ترميـز لغـة قيـود الكـائن للمجموعـات و عملياتهـا. يعـرض الشـكل رقـم (أ-٣) كـيفيـة التأكـد مـن خـلال الشـراكة fee أن قيمـة عضـو الصـنف Author ليسـت null، و ذلـك باستعمال التعبير: ( )self.fee->notEmpty.

لاحظ المرجع إلى self في تعبير لغة قيود الكائن في الشـكل أ- ٣. يشـير المرجع self إلى كـائنـات مـن النـوع Author، لأنـه تم ربطـه بــكـائن Author. وتستعمل الكلمة المفتاح self بشـكل شـائع عنـدما تقـوم بتحديـد السياق في تعبير لغة قيود الكائن، كما هو معروض في القسم التالي.

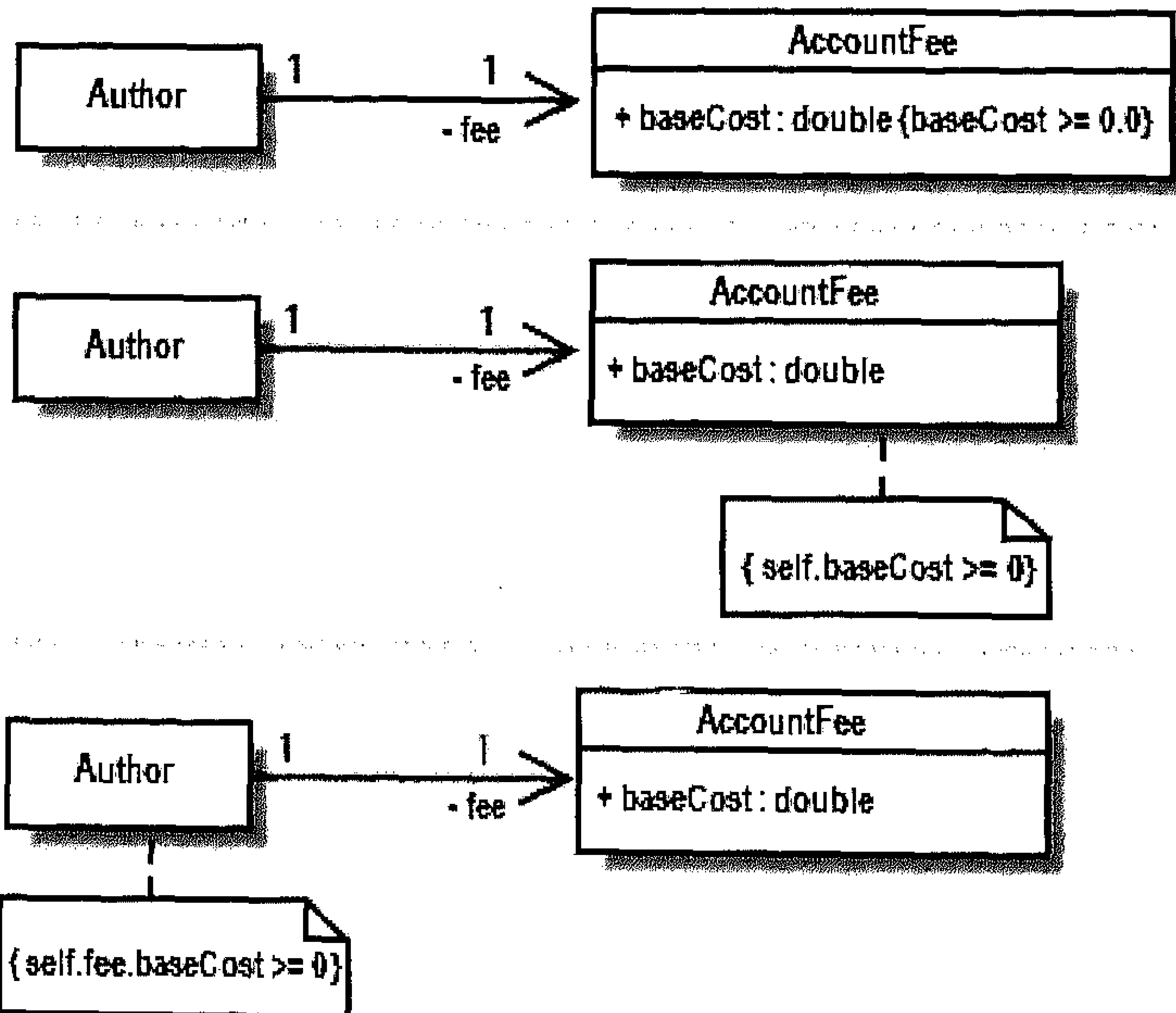


شكل رقم (أ-٣) اشتراط ألا تكون قيمة العضو **null**.

أ-٥ السياق Context

لقد تم في الشكل رقم (أ-٢) تعريف تعابير بلغة قيود الكائن على العناصر التي نريد تقيدها، بينما تم في الشكل رقم (أ-٣) تعريف تعبير OCL على الصنف الحاوي. يمكن كتابة تعبير OCL في مناطق مختلفة من المخطط. وتعتمد كيفية كتابة تعبير OCL على السياق أو الموقع المكتوبة فيه داخل المخطط.

يعرض الشكل رقم (أ-٤)، كيفية التحقق من أن الخاصية baseCost في الصنف AccountFee أكبر من أو تساوي 0 عند عدة نقاط مرجعية مختلفة في المخطط. ويعرض المخطط الأول هذا القيد في سياق الخاصية baseCost، ويعرض المخطط الثاني هذا القيد على مستوى الصنف AccountFee، كما يعرض المخطط الثالث هذا القيد على مستوى الصنف Author.



شكل رقم (أ-٤) يعتمد أسلوب كتابة القيد على النقطة المرجعية في المخطط.

إذا كانت النقطة المرجعية هي baseCost ، فيكتب القيد داخل الأقواس { } بعد التصريح عنها ، ثم نكتب التعبير: baseCost >= 0.0.

إذا كنا نشير إلى الصنف AccountFee ، فنربط ملاحظة بالصنف AccountFee ، ثم نكتب بداخلها التعبير: self.baseCost >= 0.0

أخيراً ، إذا كنا نشير إلى الصنف Author ، فنربط ملاحظة بالصنف Author ، ثم نكتب بداخلها التعبير: self.fee.baseCost >= 0.0

ويمكنك أيضاً كتابة قيود لغة قيود الكائن من دون ربطها مادياً بعناصر النموذج. على سبيل المثال ، ربما توفر أداة لغة النمذجة الموحدة المستعملة محرّر نص لإدخال القيود. إذا كانت تلك الحالة فاكتب السياق بشكل صريح. إذا كان السياق هو الصنف AccountFee ، فاكتب التالي:

Context AccountFee
inv: self.baseCost >= 0.0

تُشير الكلمة المفتاح inv إلى أن القيد هو من النوع ثابت invariant ، أو أنه يجب على شرط ما أن يبقى دائماً صحيحاً. عند تحديد السياق ، فنقوم أيضاً بتحديد نوع القيد أيضاً. ستناقش أنواع القيود في القسم "أنواع القيود" التالي.

أ-٦ أنواع القيود Types of Constraints

هناك ثلاثة أنواع من القيود:

الثوابت Invariants

الثابت عبارة عن قيد يجب أن تكون قيمته دائماً صحيحية وإلا فيكون النظام في حالة غير سليمة. ويتم تعريف الثوابت على خصائص الأصناف. وكمثال من الشكل رقم (أ-٤)، يجب على الخاصيّة baseCost في الصنف AccountFee أن تكون دائماً أكبر من أو تساوي صفراً.

الشروط السابقة Preconditions

الشرط السابق عبارة عن قيد يتم تعريفه بخصوص طريقة ما، حيث يتم التأكد من صحته قبل تنفيذ الطريقة. وتستعمل الشروط السابقة كثيراً للمصادقة على صحة بارامترات الإدخال الخاصة بالطرق.

الشروط اللاحقة Postconditions

ييتم تعريف الشرط اللاحق أيضاً على طريقة ما، و يتم التحقق من صحته بعد تنفيذ الطريقة. تستعمل الشروط اللاحقة كثيراً لوصف كيفية تغيير القيم داخل الطرق.

لقد تم تركيز الأمثلة السابقة في هذا الفصل على الثوابت، لكن يمكن التعبير عن الأنواع الثلاثة للقيود في مخططات UML أو التوثيقات المتعلقة بها. ستعرض الأمثلة التالية كيفية توفير الشروط السابقة و اللاحقة للطريقة incrementRating. لقد تم عرض مخطط الصنف المرجع للأمثلة في الشكل رقم (أ-٥).

```
BlogEntry
- rating : int
+ incrementRating(amount:int) : void
+ decrementRating(amount:int): void
+ getRating():int
```

شكل رقم (أ-٥) ستوفر الطريقة incrementRating بشروط سابقة ولاحقة.

افترض أن الطريقة incrementRating ستقوم بإضافة قيمة البارامتر amount على قيمة الخاصية rating. نريد أولاً تحديد شرط سابق مفاده أن قيمة amount أقل من كمية قانونية قصوى، لنقول القيمة ١٠٠ ، و تحديد شرط لاحق يضمن أنه قد تمت إضافة قيمة البارامتر amount على قيمة الخاصية rating. لكتابة هذه الشروط السابقة و اللاحقة ، يتم أولاً تحديد السياق و من ثم كتابة القيود.

```
context BlogEntry::incrementRating(amount: int)
pre: amount <= 100
post: rating = rating@pre + amount
```

لاحظ التوجيهة pre@ حيث إن قيمة التركيبة rating@pre هي قيمة الخاصية rating قبل تنفيذ الطريقة. ويمكن استعمال الترميز pre@ مع الطرق أيضاً:

```
context BlogEntry::incrementRating(amount: int)
pre: amount <= 100
post: getRating( ) = getRating@pre( ) + amount
```

المقصود بالتركيبة getRating@pre() هو نتيجة استدعاء الطريقة getRating() قبل تنفيذ الطريقة incrementRating().

تشكل الثوابت و الشروط السابقة و اللاحقة جزءاً من منهجية معروفة تسمى "التصميم بالتعاقد Design by Contract" التي طورها

Bertrand Meyer. وتحاول هذه المنهجية جعل الشفرة أكثر موثوقية بتأسيس عقد بين الصنف وزبائنه (أي مستخدميه). ويُخبر عقد الصنف مستخدميه أنهم إذا قاموا باستدعاء طرقه مع قيم سليمة، فسيستلمون منها نتائج سليمة. ويؤسّس العقد أيضاً ثوابت على الصنف، مما يعني أن خصائص الصنف لن تقوم أبداً بانتهاك بعض القيود.

إذا كنت تتساءل لماذا لم تصادف الثوابت والشروط السابقة واللاحقة في الشفرة، عليك معرفة أن دعم منهجية التصميم بالتعاقد يختلف من لغة برمجة إلى أخرى. وتم بناء هذه المنهجية في لغة البرمجة Eiffel التي طورها بيرتراند ميَر. تضم لغة Eiffel كلمات مفتاح خاصة بالثوابت والشروط السابقة واللاحقة، كما يتم رمي استثناء عند انتهاك أي من تلك القيود. مع لغات البرمجة الأخرى، ويجب إدارة برمجة القيود بنفسك أو باستعمال حزمة خاصة بذلك، كما تقوم به الحزمة iContract مع لغة جافا. تشكل الحزمة iContract معالج قبلي preprocessor له سمات وثائقية الطراز doc-style tags لتحديد الثوابت و الشروط السابقة و اللاحقة. تقوم iContract أيضاً برمي استثناء عند انتهاك قيد ما.

أ-٧ أتمتة لغة قيود الكائن OCL Automation

تستمد لغة قيود الكائن قوتها الحقيقية من الأدوات التي يمكنها استعمال قيودها الموجودة في نموذج UML، وذلك للتحقق من تلك القيود. بينما يوجد اختلاف واسع (في الوقت الحاضر) في نضوج الأدوات ومستوى تكاملها، إلا أن الهدف النهائي هو تحسين تكامل نموذج UML مع سلوك وقت تشغيل النظام. ويفيد ذلك بالسماح بالالتقاط المبكّر للأخطاء وبالتوفير في وقت تصحيح الأخطاء debugging.

تُركّز بعض أدوات UML على وضع قيود لغة قيود الكائن التي في المخططات داخل الشفرة المولّدة، وذلك للتمكن من التحقق من القيود في وقت التشغيل (رغم أنه حالياً، كل ما يمكننا هو اقتراح ذلك فقط أو برمجته جزئيا). وكمثال على ذلك، المنهجيات المتضمنة توليد تعليمات assert (أي التأكيد على) مباشرة في شفرتك للسماح بالتحقق من القيود، أو يمكن زخرفة شفرة البرنامج بتعليقات جافا، أو أوسمة نمط الوثائق المحتوية لقيود لغة قيود الكائن، والتي يمكن استعمالها بعد ذلك من قبل أدوات لغة قيود الكائن القياسية التي بإمكانها التحقق من القيود في وقت التشغيل. على سبيل المثال، إن أداة UML المفتوحة المصدر ArgoUML تقوم بإدراج قيود لغة قيود الكائن في شفرة جافا المولدة على شكل أوسمة نمط الوثائق. مع الأوسمة أو التعليقات في الشفرة، يمكن استغلال أدوات لغة قيود الكائن (مثل ocl4java أو صندوق أدوات Dresden للغة قيود الكائن)، التي تُحسن في الشفرة لتوفير معلومات وقت التشغيل حول انتهاكات القيود في الشفرة المُنفّذة.

كن حذراً عند التطوير في هذا المجال؛ بما أن MDA ولغة النمذجة الموحدة قابلة التنفيذ (مقدمة في الفصل الأول) تصبح بشكل متصاعد مركزية بخصوص لغة النمذجة الموحدة، يمكن حتى توقّع دمج الكثير من هذه القدرات مع أدوات النمذجة.

ملحق (ب)

تكييف لغة النمذجة الموحدة: المظاهر

ADAPTING UML: PROFILES

لقد قام هذا الكتاب باستعمال أمثلة بشفرة جافا لعرض مفاهيم لغة النمذجة الموحدة، ولكن يمكن تطبيق عناصر نموذج لغة النمذجة الموحدة المعروضة على أي نظام كائني التوجه تقريباً، وذلك بغض النظر عن لغة البرمجة (جافا أو C++ أو Smalltalk) أو عن منصة العمل المستهدفة (J2EE أو .NET) أو عن المجال الذي تعمل فيه (طبي أو فضائي).

وتشترك الأنظمة كائنية التوجه في العديد من الخصائص الشائعة من الناحية الهيكلية أو من الناحية السلوكية: حيث لديها الأصناف والتفاعلات بين الأصناف، إلى آخره. ولكن عندما يتعلق الأمر بمنصات العمل ومجالات التطبيق، فعادة ما يكون للأنظمة كائنية التوجه العديد من الاختلافات في المصطلحات. على سبيل المثال، لدى المنصة J2EE الأمور EJBs و JARs و JSPs، بينما يكون ADOs، المجمّعات assemblies و ASP لدى المنصة .NET

عندما تقوم بإنشاء نموذج لغة النمذجة الموحدة، من المفيد عنونة عناصر النموذج باستعمال المصطلحات الخاصة بالبيئة أو المنصة المختارة. بكلمة أخرى، أليس من الأفضل أن نكون قادرين على تحديد أن مكوّناً ما سيكون في الحقيقة EJB، وذلك بدلاً من تسميته مكوّن فقط؟

ستكون محاولة جعل لغة النمذجة الموحدة تستهدف كل منصة أو مجال محتمل معركة خاسرة، وليست في الواقع من روح لغة نمذجة عامة الغاية. وتدرك المجموعة OMG القيّمة على لغة النمذجة الموحدة هذا الأمر، حيث قامت ببناء آلية يمكن من خلالها تكييف لغة النمذجة الموحدة لتلائم مع حاجاتك الخاصة. تسمى هذه الآلية المظهر profile.

ب-١ ما هو المظهر؟ What Is a Profile?

تشكل المظاهر وسيلة متواضعة لتكييف لغة النمذجة الموحدة كي تناسب منصة محددة مثل J2EE أو NET، أو كي تناسب مجال محدد مثل المجال الطبي أو المالي أو الفضائي. تتكون المظاهر من الحاشيات stereotypes والقيم الملحقة tagged values ومجموعة محددة من القيود. وتشمل المظاهر المفردات شائعة الاستعمال ضمن مجال أو منصة ما، وتسمح بتطبيق تلك المفردات على عناصر النموذج لجعلها أكثر تعبيراً.

من ناحية أخرى، يمكن لأدوات توليد الشفرة استعمال المظاهر لتوليد أدوات اصطناعية معيّنة خاصة بمنصة أو ببيئة ما. كما يمكن تحويل مكوّن معنوَن بالحاشية EJB إلى الأصناف والواجهات التي يأخذها من أجل إنجاز EJB ما.

ويمكن إنشاء الحاشيات بسرعة فائقة في النسخ السابقة من UML. هذا ما أدى إلى الغموض حول معرفة متى علينا استعمال الحاشيات، حيث تم ترك المنمذجين للقيام ببناء المعايير بشكل غير رسمي وبإعادة استعمال مجموعة شائعة من الحاشيات. ولقد قامت UML 2.0 بحل هذه المشكلة من خلال الإعلان عن وجوب إنشاء تلك الحاشيات والقيم الملحقة في مظهر ما (انظر إلى القسم "القيم الملحقة" لاحقاً في هذا الفصل).

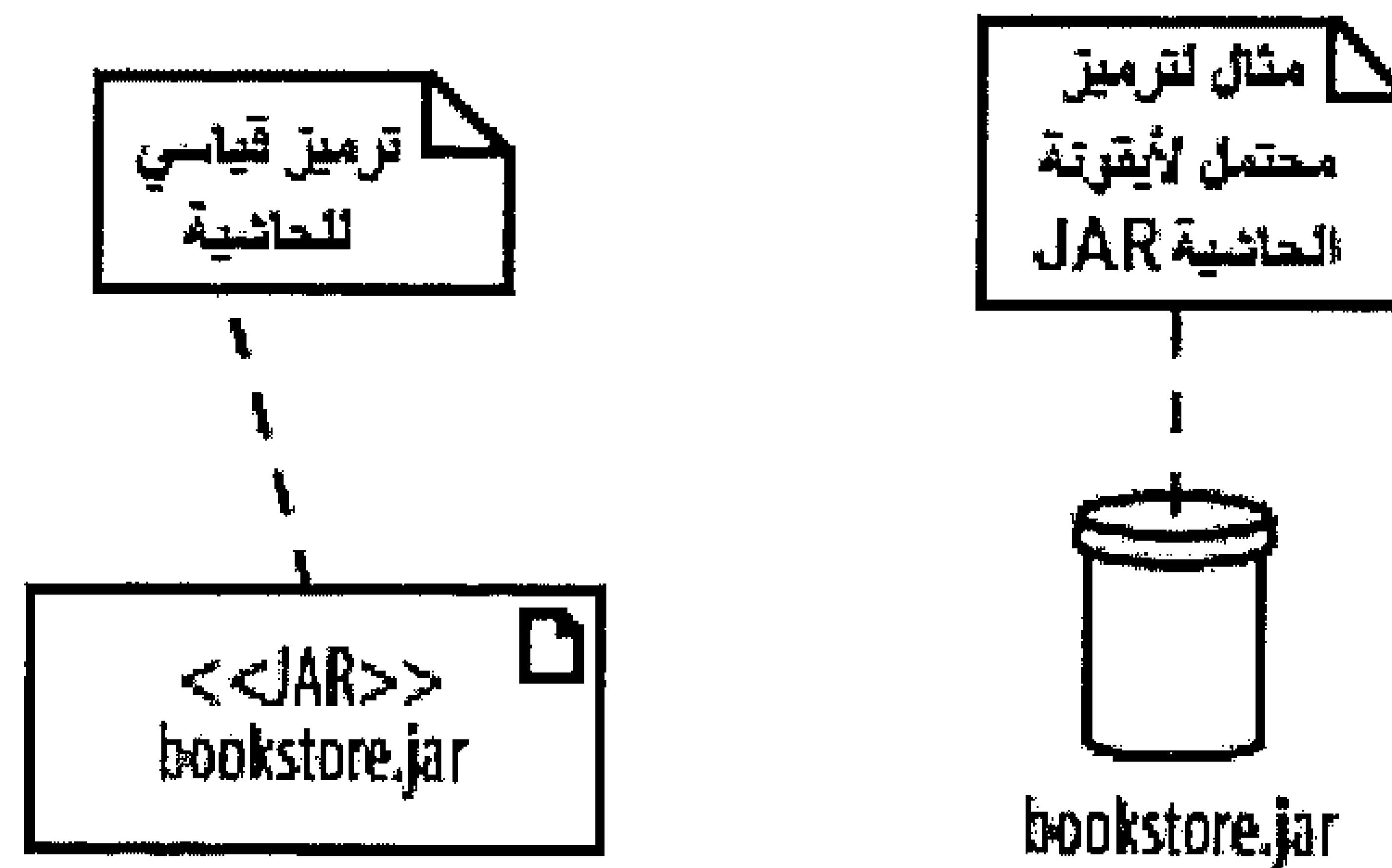
ب-٢ الحاشيات Stereotypes

تشير الحاشيات إلى استعمال أو قصد أمر ما بخصوص عنصر محدد. وغالباً ما يتم عرض الحاشيات بتحديد اسمها بين الرمزين « و »، كما في «stereaotype_name»؛ ويمكن استبدال هذين الرمزَين بالرموز << و >> إذا لم يكن هذان الرمزان متوفرَين في النظام المستعمل، كما هو معروض في الشكل رقم (ب-١).



شكل رقم (ب-١) أداة اصطناعية مُطبّق عليها الحاشية JAR؛ قد ترى هذه الحاشية في مظهر J2EE.

إذا كان للحاشية أيقونة مرتبطة بها، فقد تقوم أيضاً بعرض العنصر مع أيقونته. وعادة ما تسمح أدوات لغة النمذجة الموحدة بالتحول بين تلك الخيارات في العرض. ويعرض الشكل رقم (ب-٢) ترميز عرض الحاشية JAR القياسي بالإضافة إلى مثال عن أيقونة JAR.



شكل رقم (ب-٢) استعمال الحاشية <<JAR>> وأيقونة JAR.

لا يوجد حدّ معين لعدد الحاشيات التي يمكن تطبيقها على عنصر ما، كما هو معروض في الشكل رقم (ب-٣).

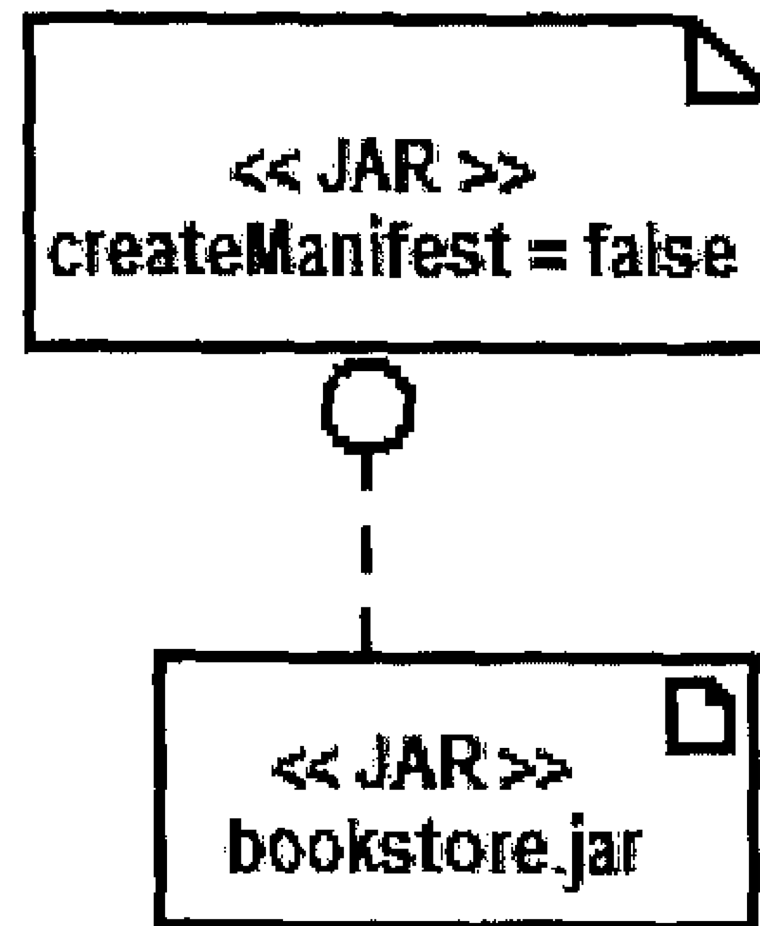


شكل رقم (ب-٣) تطبيق الحاشيتين JAR و file على الملف bookstore.jar.

ب-٣ القِيَم الملحقة Tagged Values

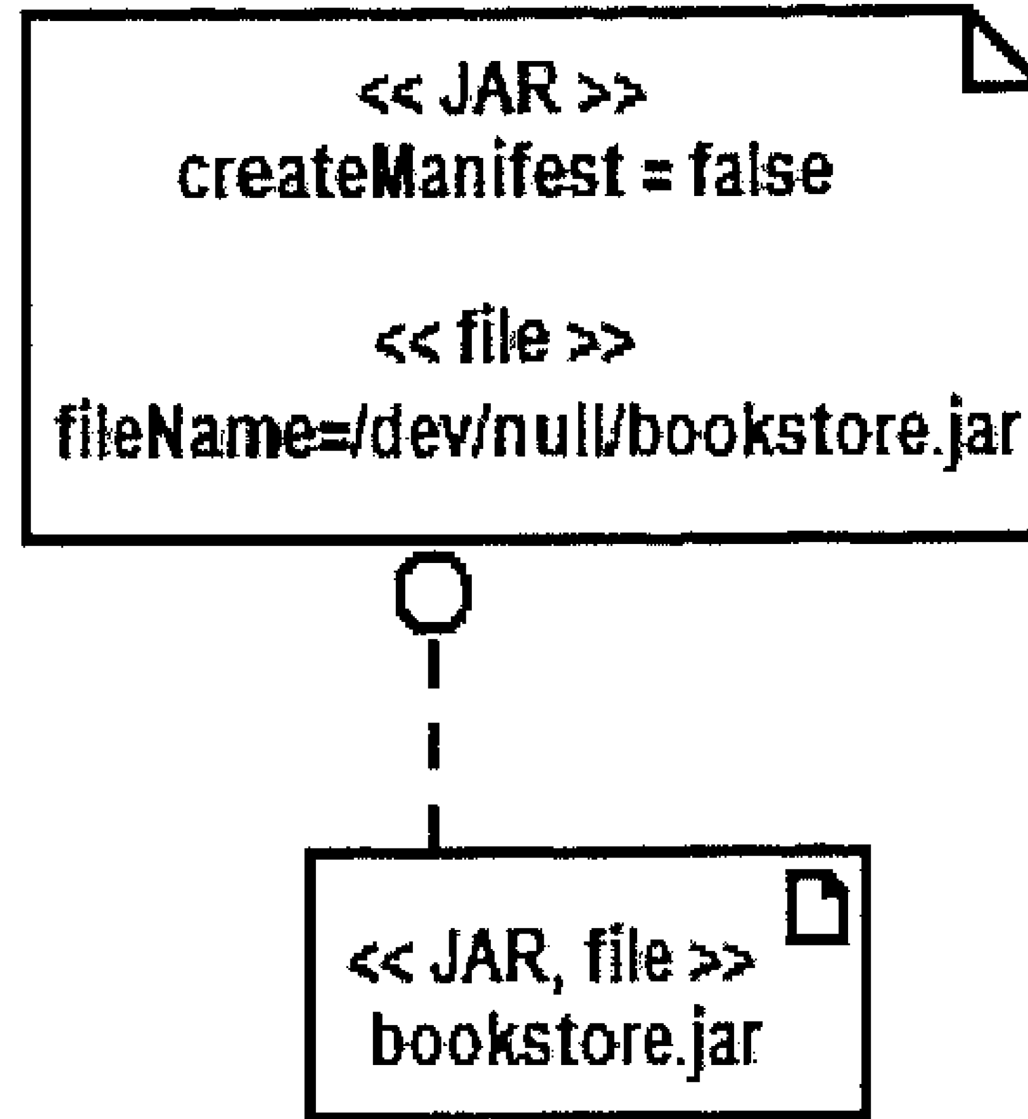
يمكن أن يكون للحاشيات قيمة ملحقة واحدة أو أكثر تابعة لها. وتقوم القيم الملحقة بتوفير معلومات إضافية مرتبطة بالحاشية.

يتم ربط العنصر المحتوي على الحاشية بملاحظة تحتوي على القيم الملحقة الخاصة بها، كما هو معروض في الشكل رقم (ب-٤).



شكل رقم (ب- ٤) تقوم القيمة الملحقة داخل الملاحظة بتحديد إذا كان يجب إنشاء قائمة manifest للملف JAR.

عند تطبيق عدة حاشيات على نفس العنصر، قم بتقسيم أي قيَم ملحقة بها في الملاحظة المرافقة لها، كما هو معروض في الشكل رقم (ب-٥).



شكل رقم (ب-٥) تطبيق عدة حاشيات حيث لكل منها مجموعه خاصة من القيم الملحقة.

ب-٤ القيود Constraints

بخلاف الحاشيات و القيم الملحقة، ليس للقيود رموز تستعمل في نماذج UML. ويتم تحديد القيود في تعريف المظهر، لكنها تفرض بعض القواعد أو القيود على عناصر النموذج. ويوجد مثال معروض عن قيد ما في القسم "إنشاء المظهر".



يوجد مقدمة عن القيود خارج سياق المظاهر في الفصل الخامس.

ب-٥ إنشاء المظهر Creating a Profile

عادة ما نقوم ببساطة باستعمال مظهر موجود سابقاً (الذي يكون مدمجاً في أداة UML أو موفراً من طرف مصدر قياسي مثل OMG). على أية حال، إذا وجدت أنه لا يتوفر لديك مظهر قياسي، فستسمح لك الكثير من أدوات لغة النوذجة الموحدة إمكانية إنشاء مظهراً خاصاً بك.



كن حذراً عند إنشاء مظاهر خاصة بك، حيث تكمن القوّة الحقيقية للمظاهر فقط عندما تكون قياسية وشائعة الاستعمال، ستتم مناقشة المظاهر لاحقاً في هذا الفصل في القسم "لماذا الانزعاج مع المظاهر؟".

قد تسمح أداة لغة النمذجة الموحدة المستعملة بإنشاء مظهر محدد من خلال استعمال حوار نصي بسيط؛ على سبيل المثال، ربما تسأل عن اسم الحاشية وتطلب اختيار نوع العنصر الذي يمكن أن تُطبّق عليه. على أية حال، يشبه النموذج التخطيطي المخفي لمظهر محدد ذلك المعروض في الشكل رقم (ب-٦).

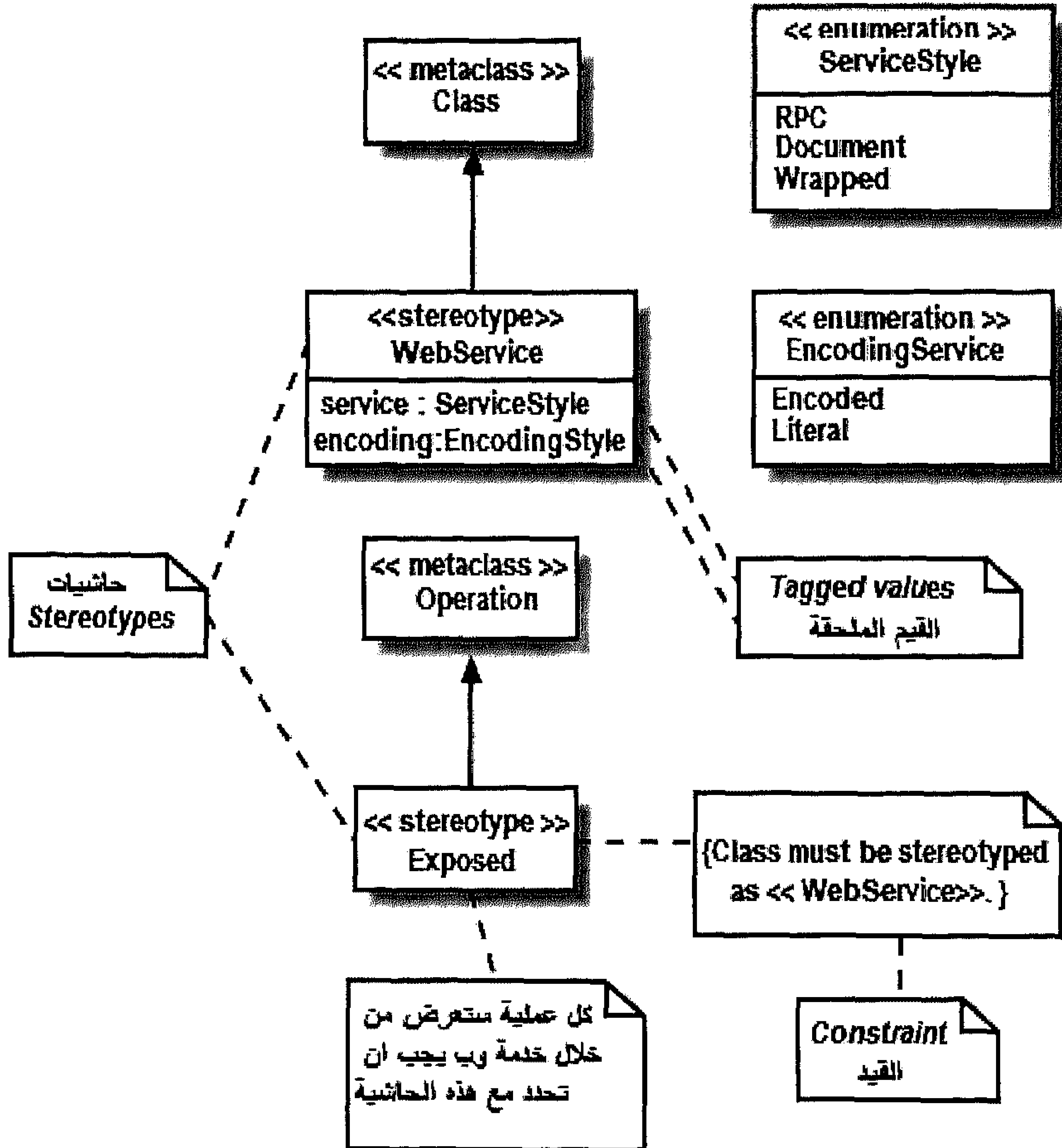
إن الحاشيات المعرّفة في المظهر هي نفسها التي قد تم إعطاؤها الحاشية القياسية <<stereotype>>. وتوجد حاشيتان جديدتان مصرّح عنهما في الشكل رقم (ب-٦): WebService و Exposed.

ولعرض إمكانية تطبيق الحاشية WebService على الأصناف، تتم الإشارة بسهم التوسّع extension من WebService إلى الصنف Class. يكون لسهم التوسّع رأس معبّأ ويقوم بربط الحاشية الجديدة بنوع العنصر الذي يمكن تطبيقها عليه. ويستعمل سهم التوسّع أيضاً لعرض إمكانية تطبيق الحاشية Exposed على العمليات Operations.

إذا كان للحاشية قيم ملحقة، فيتم تسجيلها في مقصورة تحت اسم الحاشية. ويتبع الحاشية WebService القيمتان الملحقتان خدمة service وتشفير encoding. يتم عرض القيم المحتملة لتلك القيم الملحقة في قائمتي السرد (enumerations) ServiceStyle و EncodingStyle.

أخيراً، تكون أي قيود قابلة للتطبيق على استعمال الحاشيتَين WebService و Exposed محدّدة في ملاحظات. لدى الحاشية Exposed

قيد ما (داخل الأقواس { }) ويُحدّد أنه يمكن تطبيقها فقط على عمليات الأصناف التي تكون نفسها لها حاشية WebService.



شكل رقم (ب-٦) إنشاء مظهر جديد يحتوي على الحاشيتين Exposed و WebService، وبعض القيم الملحقة و القيود التابعة لهما.

ب-٦ العمل مع نموذج النموذج
Working with the Meta-Model

عند هذه النقطة، ربما فكرت أن النموذج الذي في الشكل (ب-٦) يبدو مشابهاً لنماذج لغة النمذجة الموحدة الأخرى التي سبق ورأيتها في هذا الكتاب. على أية حال، يحتوي هذا النموذج على بعض الاختلافات البارزة: من ناحية، وترتبط الحاشية WebService بالعنصر Class، ومن ناحية أخرى، يختلف سهم التوسّع عن العلاقات التي رأيتها سابقاً. وبشكل عام، لن تقوم مطلقاً بالإشارة الصريحة إلى الصنف Class في نماذج لغة النمذجة الموحدة لأنه عنصر نموذج النموذج فيها.

ولقد تم تقديم مصطلح نموذج النموذج meta-model في الفصل الأول. وتقوم نماذج النموذج بتعريف القواعد الخاصة بكيفية عمل عناصر لغة النمذجة الموحدة، مثال على ذلك: يمكن أن يكون لصنف ما صنف فرعي أو يمكن لصنف ما المشاركة مع أي عدد من الأصناف الأخرى. عندما تقوم بنمذجة مظهر ما، فتكون تعمل مع نموذج النموذج، وتكون تكيّف قواعد لغة النمذجة الموحدة العادية لأجل سياق محدّد.

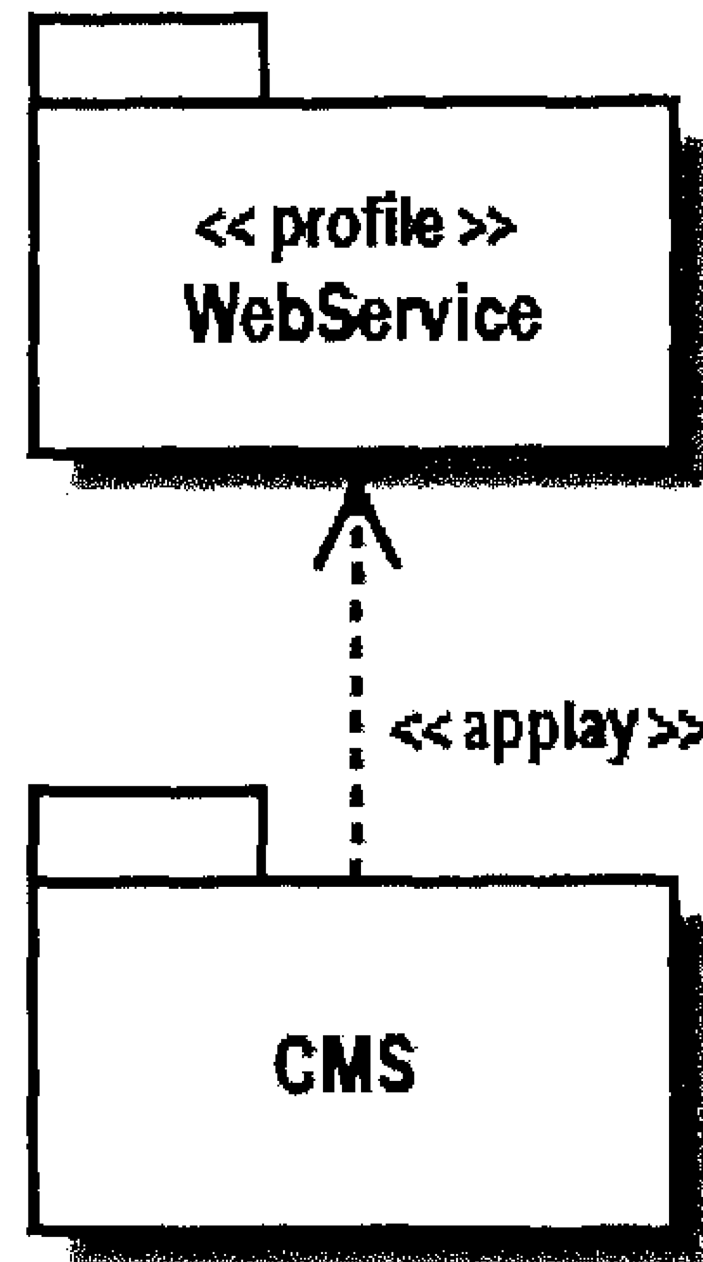


قد يبدو تكييف لغة النمذجة الموحدة لأجل سياق خاص بك خطيراً في بادئ الأمر، وكأنك تنشئ لغة خاصة بك تقريباً! لكنها ليست الحالة هنا في الحقيقة، فالمظاهر هي طريقة آمنة ومسيطر عليها لتكييف لغة النمذجة الموحدة، لكن يجب استعمالها فقط عند الحاجة إليها حقاً (انظر إلى القسم "لماذا الانزعاج مع المظاهر؟"). يمكنها أن تكون طريقة قوية لجعل نموذجك يعني أكثر بكثير مما يفعله مع لغة النمذجة الموحدة القياسية وحدها.

ب-٧ استعمال المظهر Using a Profile

يعرض النموذج في الشكل ب- ٦ كيفية إنشاء المظهر خدمة وِبْ WebService. من أجل استعمال المظهر فعلياً، فإننا نُطبّق apply المظهر على الحزمة التي ستستعمله.

من أجل تطبيق مظهر ما على حزمة معينة، نقوم برسم سهم منقط من الحزمة إلى المظهر المستعمل، و ذلك مع كتابة الحاشية <<apply>> بمحاذاة السهم، كما هو معروض في الشكل رقم (ب-٧).

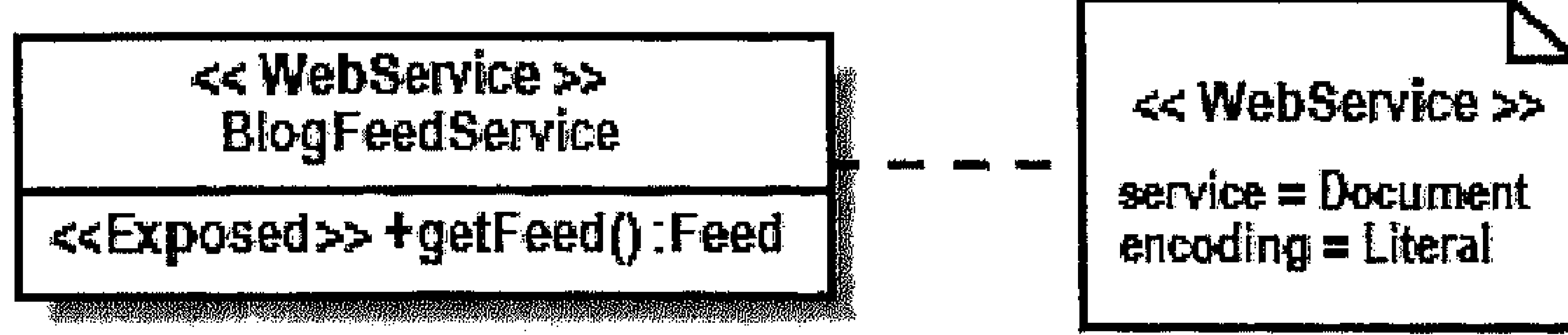


شكل رقم (ب-٧) يسمح تطبيق المظهر WebServcie على الحزمة CMS باستعماله في نموذج نظام إدارة المحتوى CMS.



> لا تستعمل كل أدوات لغة النمذجة الموحدة هذه الطريقة لتطبيق مظهر ما على النموذج. على سبيل المثال، فقد تسمح الأداة بتحديد المظهر الذي سيُطبّق على حزمة ما من خلال استعمال صندوق حوار.

بما أنك قمت بتطبيق المظهر، ربما تستعمل المظهر في نموذج نظام إدارة المحتوى CMS، كما هو معروض في الشكل رقم (ب-٨).



شكل رقم (ب-٨) تطبيق عناصر المظهر WebService على الصنف BlogFeedService في الحزمة CMS.

تم في الشكل رقم (ب-٨) تحديد الحاشية WebService للصنف BlogFeedService. كما تم تحديد الحاشية Exposed لطريقته الوحيدة، و ذلك للسماح بعرضها من خلال خدمة الويب. وقد تم ربط القيم الملحقة التابعة للحاشية WebService داخل ملاحظة، حيث تم تأهيلها بقيم من قوائم سرد enumerations.

ولم يظهر القيد الخاص بالمظهر خدمة وِبْ بشكل صريح في هذا النموذج، لكنه مُستعمل لأن الحاشية Exposed الخاصة بالعملية getFeed() موجودة في صنف يستعمل الحاشية WebService.

ب-٨ لماذا الانزعاج مع المظاهر؟

Why Bother with Profiles?

تأتي القوّة الحقيقية للمظهر عند استعماله من قِبَل العديد من الناس المهتمين بهكذا منصّة أو مجال. بالإضافة إلى تقديمه مفردات عامة، فهو يسمح أيضاً بزيادة فائدة الأدوات المولّدة لشفرة المصدر والأدوات الاصطناعية الأخرى المعتمدة على المظهر. على سبيل المثال، يمكن لأداة ما تحويل نموذج يستعمل مظهر خدمة وِبْ خاص بنا إلى خدمة وِبْ قابلة للنشر. يمكن لهكذا أداة توليد إنجاز الصنف و تأهيل ملف وصف النشر مع نوع الخدمة والقيم المشفرة، وذلك مع إبقاء النموذج متزامن مع الشفرة. كمثال آخر، توفر بيئة التطوير Omondo Eclipse IDE plug-in مظهر J2EE، أي

عند تطبيقه على النموذج، يسمح بالتوليد الآلي لخليط من الأصناف المتطلبة لإنجاز EJB (قبل EJB 3.0)، وحتى نشرهم على خادم التطبيق.

تحافظ OMG على بعض المظاهر الشائعة، مثل المظاهر المتعلقة بالاختبار و بـ CORBA. على سبيل المثال، يصف المظهر اختبار مخطط الوحدة JUnit (وحدة جافا واسعة الاستعمال لاختبار إطار العمل).

ملحق (ج)

لمحة تاريخية عن لغة النمذجة الموحدة

A HISTORY OF UML

لم تكن لغة النمذجة الموحدة أبداً لغة النمذجة الواقعية كما هي اليوم. في الحقيقة، منذ مدة ليست ببعيدة، كان كل شخص معني بنمذجة النظم المعقدة يستعمل عدداً كبيراً من لغات النمذجة المختلفة (بعضها رسمي وبعضها غير رسمي)، ولكل منها منهجية تطوير خاصة بها.

ولم تكن المشكلة جلية في أي مكان أكثر مما كانت عليه في نمذجة البرمجيات. وكان التوجه الكائني قد أصبح للتو تقنية تطوير برمجيات معترف بها كليّا، ولم تمضِ مدة طويلة حتى بدأت طرق النمذجة الجديدة بتضمين هذه التقنية الثورية في تطبيقها.

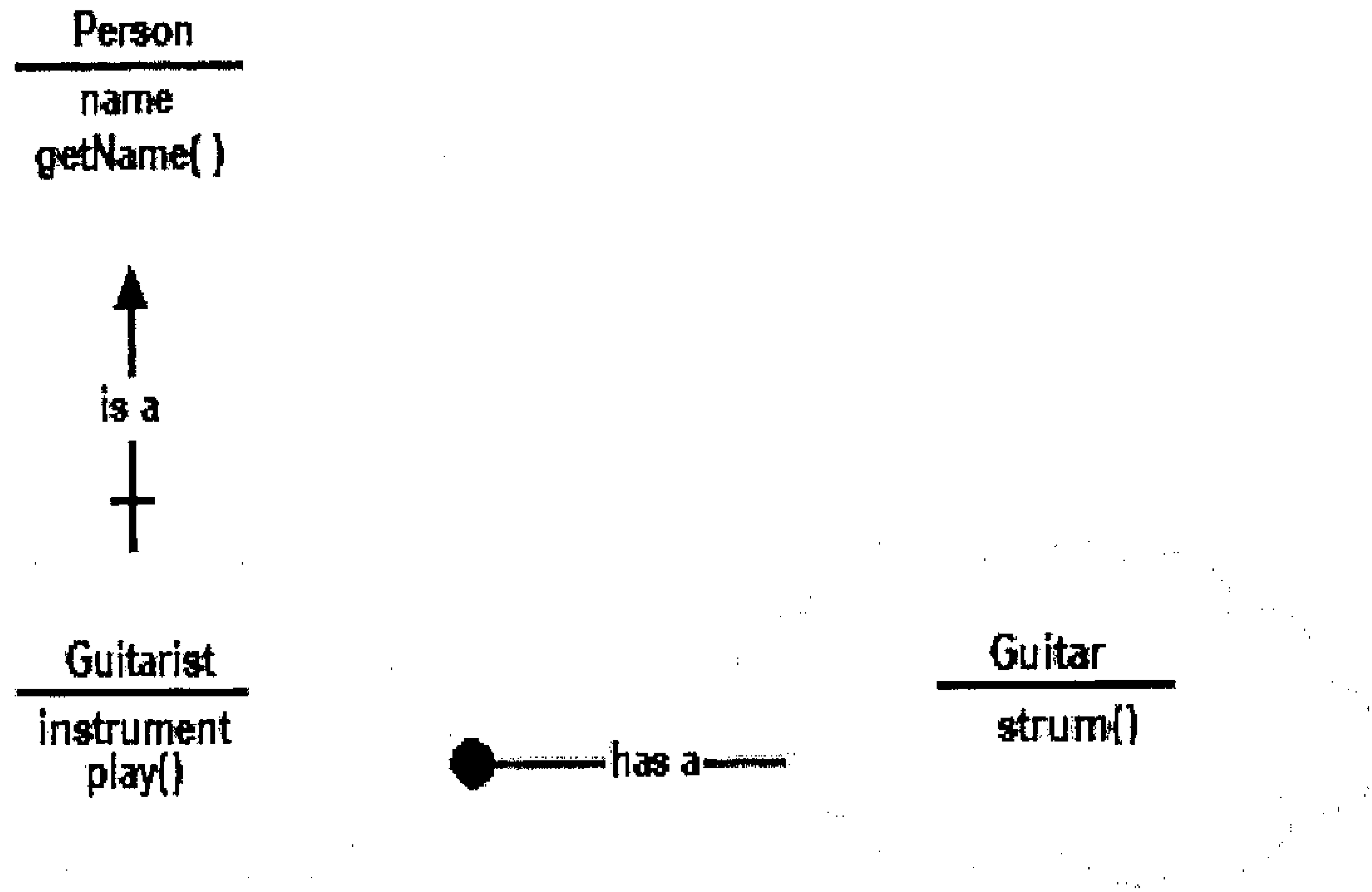
ولسوء الحظ، رغم اعتبار النمذجة ذات التوجه الكائني للبرمجيات أمراً جيداً، فقد ألغت فوضى منهجيات النمذجة المختلفة والمتعارضة كثيراً من حسناتها الواعدة. إذا كنت تصمّم باستعمال لغة نمذجة محددة وكان عضو آخر من مجموعتك يستعمل لغة نمذجة أخرى، فستفقد كلياً حسنات نقل التصاميم فيما بينكم. لقد كانت مجموعات النظام والبرامج مجبرة على اختيار لغة نمذجة محددة، مع إدراك احتمال كون اختيارهم قراراً خطيراً يمكن أن يمنع مجموعات أخرى من الانضمام بسهولة إلى الجهد المبذول في التصميم.

يشار الآن إلى زمن التشويش والفوضى في عالم نمذجة البرمجيات بشكل مثير كأنه "حروب الطرق method wars". كان هناك ثلاثة طرق أساسية من بين أبطال حروب الطرق: طريقة Grady Booch وطريقة Ivar Jacobson وطريقة James Rumbaugh. كان لكل واحد من هؤلاء المبتكرين طرق تطوير برمجيات خاصة به، ولغة نمذجة ذات ترميز مختلف. وبأهمية أكبر، فقد كانوا أيضاً على رأس جماعة المستعملين الخاصة بهم والمناصرة لمنهجيتهم في نمذجة البرامج. لقد كانت هذه المنهجيات الثلاث في تطوير البرامج و اللغات والترميزات المتعلقة بها، هي التي شكلت أساس لغة النمذجة الموحدة.

ج-١ أخذ جزء واحد من منهجية OOAD

كانت منهجية Grady Booch تدعى التحليل و التصميم كائني التوجه Object-Oriented Analysis and Design (OOAD)، أو طريقة Booch بشكل عريض. لقد انطوت هذه العناوين الكبيرة على طريقة تتضمن لغة نمذجة مبنية من مخططات تعرض الأصناف والحالة وحالة الانتقالات والتفاعلات والوحدات وسير عمليات.

ربما كانت هذه المجموعة الهائلة من المخططات معروفة بشكل أفضل من خلال الترميز الخاص بالصنف، والذي كان كالغيوم مع مجموعة أسهم ذات أسماء بسيطة مثل لها **has a**، والتي يمكن استعمالها لتحديد أنواع مختلفة من العلاقات بين الأصناف، كما هو معروض في الشكل رقم (ج-١).



شكل رقم (ج-١) ترميز الغيمة في OOAD الذي يصف الأصناف و العلاقات التي بينها.

لقد احتل أسلوب الترميز بالغيمة والتسمية البسيطة لأسهم العلاقات بين الأصناف مكاناً في قلوب متبنيها لدرجة استغراقهم في الذكريات عنها حتى يومنا هذا. في الحقيقة، لقد أثار استعمال كل من ترميز الغيمة وترميز المستطيل للأصناف بعضاً من الحجج الأقوى، وعديمة الفائدة، أثناء بداية لغة النمذجة الموحدة.

ج-٢ مع قليل من OOSE

ربما كان Ivar Jacobson ومنهجيته في الهندسة البرمجية الكائنية التوجه Object-Oriented Software Engineering (OOSE) معروفاً جيداً بسبب التقنية الثورية في إسقاط المتطلبات نحو تصاميم كائنية التوجه للبرمجيات، والمسماة حالات الاستخدام. كان هناك تركيز واضح في OOSE على أسر مجال المشكلة ونمذجته بشكل

دقيق، وقد كانت حالات الاستخدام تقنية أساسية في إنجاز ذلك كجزء من نموذج المتطلبات.

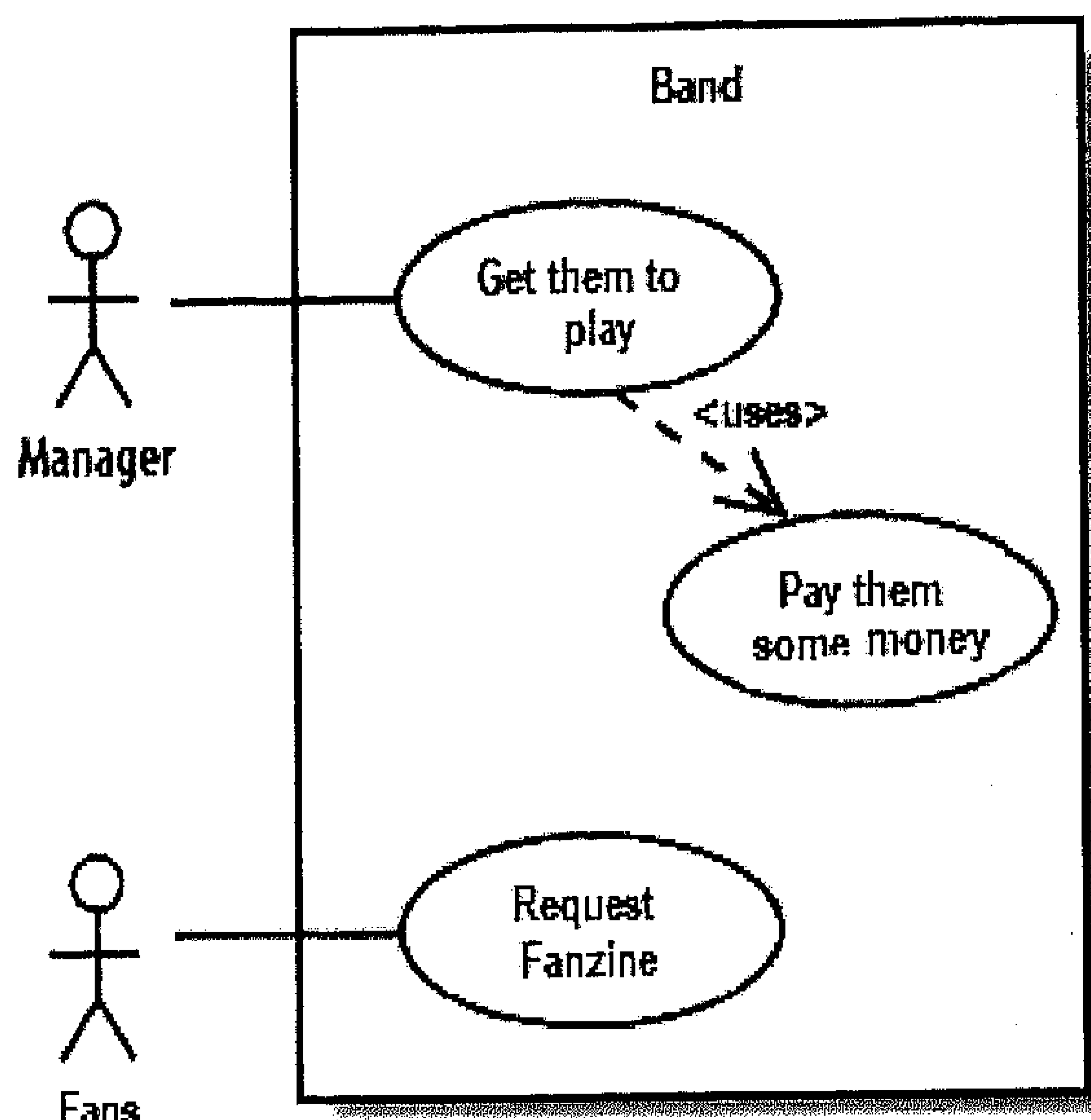
لم تكن OOSE خاصة بالمتطلبات فقط، ومع ذلك كان عندها نماذج مقابلة للتحليل والتصميم أيضاً. لقد كان نموذج التحليل في OOSE من الأمور المستعملة في نمذجة علاقات الكائن. وكان يميّز بين كائنات الكينونة entity التي تحتوي على بيانات، وكائنات التحكم التي تتحكم بكائنات أخرى، وكائنات الواجهة التي تتفاعل مع المستخدمين أو الأنظمة الخارجية الأخرى. لقد ورثت لغة النمذجة الموحدة هذه الأنواع من الكائنات، وهي شائعة بشكل خاص مع مخططات التتابع (انظر إلى الفصل السابع).

ويسمح نموذج التصميم بوصف كيفية تصرف النظام باستعمال مخططات الحالة والانتقال ومخططات التفاعل، والتي ما زالت حاضرة إلى حدّ ما في لغة النمذجة الموحدة اليوم (رغم التغيير في الترميز). لقد تم تغطية مخططات الاتصالات في الفصل الثامن؛ وتم تغطية مخططات الحالة والانتقال في الفصل الرابع عشر.

ولتكملة الوصف، لقد قامت OOSE أيضاً بتحديد نماذج الإنجاز والاختبار. يقوم نموذج الإنجاز بأسر كيفية إسقاط حالات الاستخدام على إنجاز النظام المعني، وينهي نظام الاختبار الحلقة بعرض كيف تستطيع حالات الاستخدام قيادة تطوير اختبارات نظام ما.

وبالرغم من كل هذه النماذج المختلفة، تعتبر حالات الاستخدام الأمر الذي اشتهرت به OOSE؛ انظر إلى الشكل رقم (ج-٢) كمثال عنها. لقد مرّت لغة النمذجة الموحدة بتغيرات مختلفة على مرّ السنين، لكن

حالات الاستخدام هي التقنية الوحيدة التي لم تتغير مع كل تركيباتها.

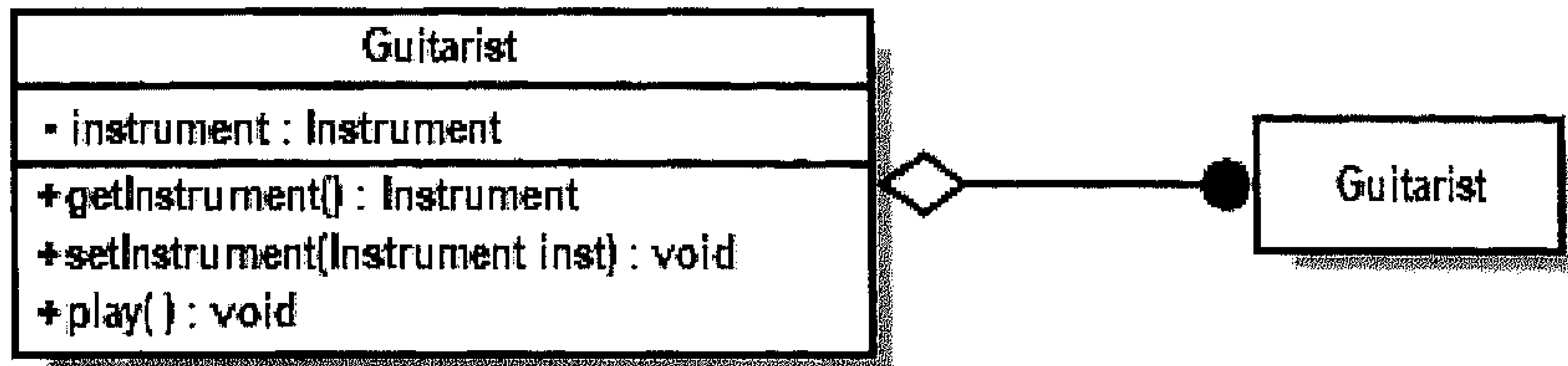


شكل رقم (ج-٢) يصف ترميز حالة الاستخدام في OOSE ثلاث حالات استعمال داخل النظام ومؤثرَين خارجَين (مستخدمان) يتفاعلان مع تلك الحالات.

ج-٣ إضافة قدر قليل من OMT

إذا قدمت طريقة OOAD (Booch) نمذجة الأصناف، وقدمت طريقة OOSE (Jacobson) نمذجة حالات الاستخدام، والحالة والانتقالات والتفاعلات، ثم أضاف James Rumbaugh تقنيته في نمذجة الكائن (OMT) Object Modeling Technique. رغم أن مخططات Booch للأصناف والكائنات كانت محبوبة كثيراً من قِبَل مُتبنّيها (المستخدمة للغيوم)، فقد كان ترميز مخطط الأصناف والكائنات التابع للطريقة OMT الأكثر تأثيراً وتم اختياره. لقد كان ترميز OMT لمعاينة الكائنات

أسهل في الرسم، حتى إذا لم تكن العلاقات بين الأصناف بديهية كما كانت مع Booch (انظر إلى الشكل رقم ج-٣).



شكل رقم (ج-٣) هذا المخطط سهل القراءة حتى بالنسبة للمطور المبتدئ.

هذا لا يعني أن ترميز الصنف والكائن هو كل ما أضافته OMT إلى الخليط. فقد كان لدى OMT ترميزاً للمخططات أيضا يعرض الخصائص الديناميكية للبرمجيات، والتي تسمى مخططات التتابع.

ج-٤ تحضير من ١٠ إلى ١٥ سنة

ليس القول بأن عالم النمذجة كان في فوضى هو استهانة بالأمر (فقد كانت حرباً بالنهاية!). وربما كان مجرّد اقتراح منهجية موحدة للنمذجة يجذب الاعتراضات العاطفية كحماية مهارات وأدوات ومنهجية ممارسي النمذجة.

لكنه حان الوقت لحصول تغيير ما. في أوائل ١٩٩٦، أجرى Jacobson كالعادة اتصالاً هاتفياً مع Richard Mark Soley في بيته، في ساعة متأخرة من الليل ، وأكد له أن الوقت مؤآت لتوحيد المعايير للغة نمذجة ، ووضع حد لحروب طرق النمذجة. عند هذه النقطة، كان إجمالي قيمة أدوات النمذجة في السوق على نطاق عالمي يساوي حوالي ٣٠ مليون دولار، حيث كانت مقسّمة بين عدة باعة. وقد كان البائع الأكبر هي

شـركـة Rational Software Corporation، لـكـن حتـى عنـد هـذه النقطـة، كـانت هـذه الشـركـة تعتبر كسمكة صغيرة لها ٢٥ مليون دولار مقارنة بعمالقة البرامج الهائلة مثل Microsoft و IBM.

وكخطـوة أولى، أنشأ Soley و Jacobson قائمـة تضـم كـلا المنهجيين الرئيسيين في المجال، وتمت دعوتهم لحضور اجتماع في OMG لاستكشاف إمكانية تطوير معيار قياسي. وكان القليل منهم فقط يعرف كم سيكون ذلك الاجتماع مهماً وناجحاً. كانت OMG تقليدياً هيئة مقاييس موجهة بشكل محدد نحو الأنظمة الموزعة؛ على أية حال، ضمن عمل مبدع وحكيم، نظّمت OMG اجتماعها الأول الهادف إلى إنشاء لغة نمذجة قياسية بضيافة شركة Tandem Computer في مدينة San Jose بكاليفورنيا، وقد حضر تقريباً كل منهجيّ رئيسي أو ممثل عنه ذلك الاجتماع. ووفقاً للأسطورة، كان المنظمون حريصين جداً في ترك كل النوافذ مفتوحة كي لا تنفجر الغرفة من عدد المتبجحين فيها، وقد كان الاجتماع رائعاً جداً. تم الإدراك باكراً أن الجانب الأكثر صعوبة من الاجتماع كان على الأرجح إيجاد الشخص المناسب لترأسه. فقد احتاجوا إلى شخص معروف كمنهجي ويكون أيضاً نزيهاً ومركزياً بما فيه الكفاية، ويمكنه توجيه الاجتماع فعلياً نحو خاتمة مفيدة.

لقد كانت ماري لومز Mary Loomis ذلك الشخص. في تلك الأيام، كانت ماري مديرة باحثة في Hewlett-Packard، وكانت عضواً في فريق لدى جينيرال إلكتريك المطورة لمنهجية OMT. كانت ماري الشخص المثالي لإبقاء كل أولئك المتبجحين تحت المراقبة. وبسرعة كبيرة، استطاعت ماري إدارة كل الخبراء الموجودين في الغرفة لإحراز تقدم نحو اتفاقية ما. لم يكن الهدف عبارة عن مناقشة تقنية حول أي التقنيات سيتم

استعمالها، أو ما إذا كان سيرسم الصنف على شكل صندوق أو غيمة، لكنه كان لتحديد كيفية تطوير لغة نمذجة موحدة.

هذا الذي أتت به OMG إلى الخليط. فقد برعت OMG في جعل المتنافسين المباشرين يتفقون بخصوص القضية المطروحة، والتي كانت السمة الأكثر أهمية للوصول إلى لغة نمذجة موحدة.

وافق المشاركون على أنه:

- حان الوقت لإيجاد معيار قياسي.
- سيحاولون استعمال عمليات OMG القياسية لتطوير ذلك المعيار القياسي.

كانت تلك الأهداف البسيطة إنجازاً مدهشاً حقاً. عند تلك النقطة من الوقت، كانت OMG قد استعملت فقط عملياتها القياسية لتطوير كائن حاسوبي قياسي موزّع، مثل CORBA وخدماته. لم تقم OMG أبداً بإنشاء أي شيء مثل تطوير معايير قياسية، كالتي يجب أن تكون عليها مواصفات لغة النمذجة الموحدة. من دون أدنى شك، إن إدارة مجتمع ملهم كالمنهجيّين، وعلى رأس ذلك، بناء معيار قياسي ناجح يمكن لأي شخص التسجيل فيه ليكون أرضية جديدة وخطيرة من أجل OMG.

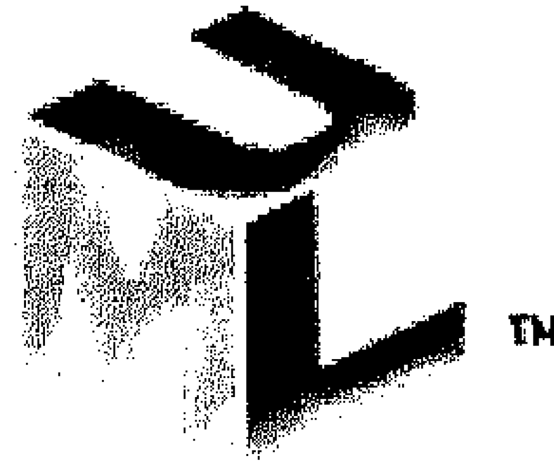
لكن كانت OMG وسط انتقال ما. أدركت أثناء ذلك الانتقال أن أحد أكبر قدرات المجموعة كان عملياتها القياسية نفسها وليست أي تقنية معينة. بمتابعة طريقة عملها، قامت OMG بتطوير وثيقة متطلبات وإرسالها إلى الصناعيين حيث تصف ما هو متطلب بالضبط من لغة النمذجة الموحدة. كان بالتالي على الصناعيين إرسال أفكارهم الخاصة عن كيفية تلبيتهم تلك المتطلبات.

في منتصف ١٩٩٧، استلمت OMG ما كان سيصبح اقتراحاً مشتركاً مقبولاً للغة النمذجة القياسية. وكان هذا الاقتراح المشترك، المكتوب من قِبَل ٢١ شركة مختلفة، ثمرة دمج كل الاقتراحات الخاصة بتلك الشركات. لقد انتهت العملية الكاملة في سبتمبر/أيلول ١٩٩٧ عندما نشرت OMG مواصفات لغة النمذجة القياسية، لكن كان هناك مشكلة سطحية باختيار الاسم المُفضّل لها. لقد قررت OMG تسمية لغة النمذجة القياسية بلغة النمذجة الموحّدة، لكن كانت الشركة Rational Software Corporation التي وافقت على المقترح المشترك الأولي تمتلك الاسم التجاري "لغة النمذجة الموحدة" .

وقامت شركة Rational Software بتوظيف كل من Jacobson وBooch و Rumbaugh بشكل جماعي (المعروفين بالأصدقاء الثلاثة)، وكانت قد قدمت كمية ضخمة من المساهمات في تطوير معيار OMG القياسي، بالإضافة إلى الاستمرار بالبحث نحو مواصفات مشتركة خاصة بها من أجل لغة نمذجة ما. لقد جمعت لغة النمذجة الخاصة بشركة Rational منهجيات وأدوات الأصدقاء الثلاث الشعبية، لكن لسوء الحظ، فقد تم أيضاً تسميتها لغة النمذجة الموحّدة.

لقد كان وقتاً حاسماً؛ هل سيعود الصناعيون للتشويش على لغة النمذجة الموحدة الخاصة بالشركتين OMG و Rational معاً، أو هل على OMG إيجاد اسم جديد كلياً وتفقد بذلك أي اعتراف بالاسم كانت قد اكتسبته سابقاً العلامة التجارية UML؟ لحسن الحظ، فقد كانت هناك نهاية سعيدة جداً لهذه القصة. من أجل حل كابوس التسمية، قامت OMG بإنجاز موفق.

ورغم وجود بعض القيمة التسويقية الهامة للعلامة التجارية UML، فقد استطاعت OMG إقناع Rational بالتبرّع دون مقابل بالاسم UML وشعار المكعّب معاً (انظر إلى الشكل رقم (ج-٤)). من هنا، يمكن أن تتقدم OMG بنجاح مع لغة نمذجة قياسية مفتوحة حقاً، والتي استطاعت تسميتها رسمياً لغة النمذجة الموحدة UML.

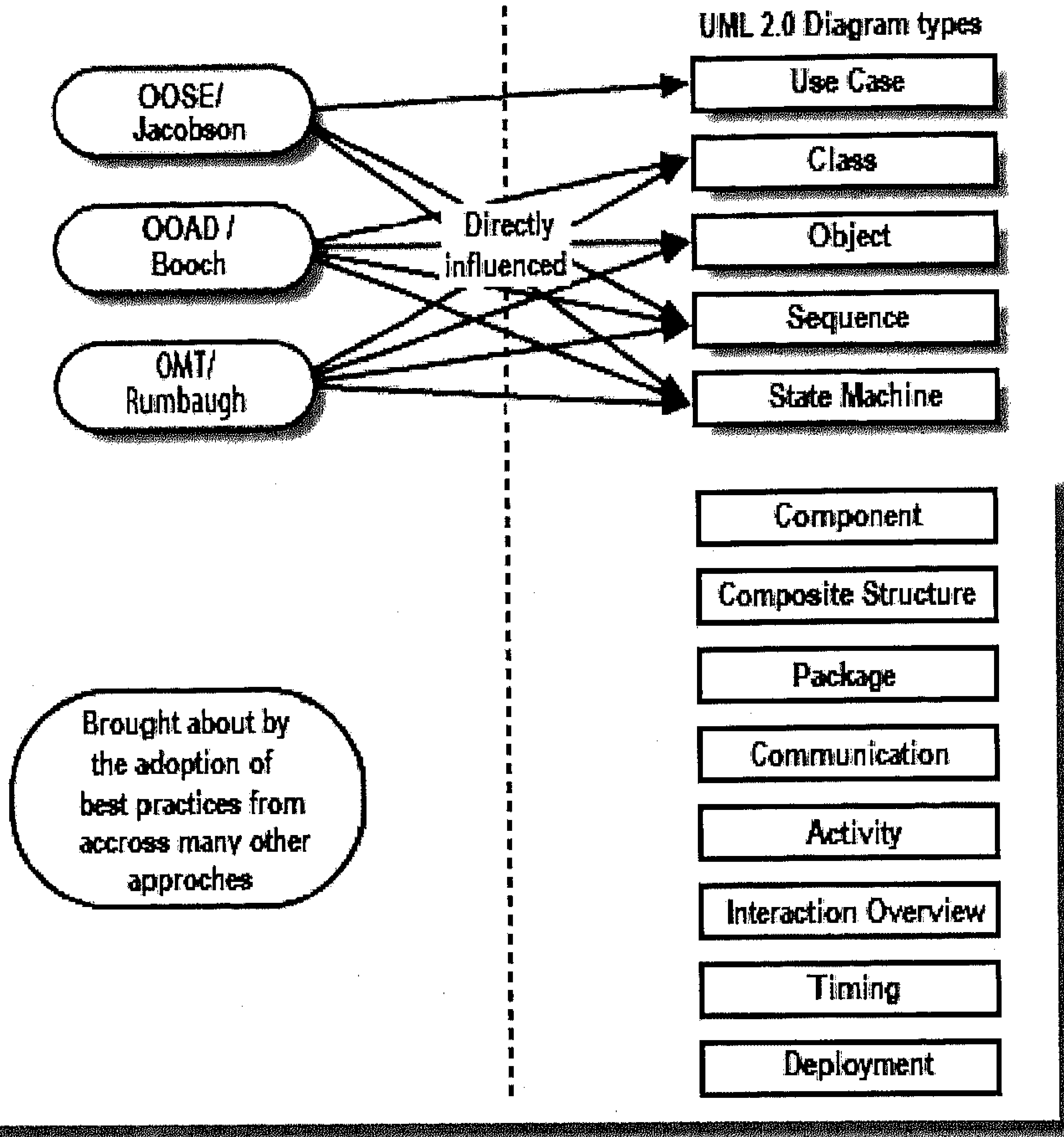


شكل رقم (ج-٤) شعار المكعب الخاص بلغة UML.

بعد ذلك بسنتَين، اعتقد الناس أنه كانت Rational وحدها المعنية بتطوير مواصفات لغة النمذجة الموحدة، وذلك بسبب نشأة اسم وشعار UML في Rational، وحيث كانت أداة Rational Rose هي أداة النمذجة الأكثر شعبية في ذلك الوقت. في الحقيقة، لم ترد بعض الشركات أن تسمى لغة النمذجة القياسية بالاسم UML بسبب اعتقادهم أن الناس سيستمرون بربط اسم UML بشركة Rational. وقد ثبت مع مرور الوقت أن تلك المخاوف كانت خاطئة بصورة عامة، ويعترف الآن أكثر من ٩٠ بالمائة من الممارسين في المجال بأن UML هي معيار قياسي تملكه وتشرف عليه الشركة OMG.

لقد خضعت لغة النمذجة الموحدة لعدة تنقيحات كما تم تطويرها تدريجياً لتلائم تنوّع التقدم الصناعي الجديد والتقنيات الأفضل تطبيقاً. تبقى المساهمة الأصلية من قِبَل Jacobson و Booch و Rumbaugh مهمة

جداً و تعمل حتى الآن بنجاح إلى جانب المجموعة الكاملة المحتملة لمخططات UML 2.0، كما هو معروض في الشكل رقم (ج-٥).



شكل رقم (ج-٥) لقد تم بناء لغة النمذجة الموحدة على أفضل الممارسات السابقة، كما أنها تستمد تقنيات متعددة من OOSE و OOAD وOMT بالإضافة إلى الكثير من التقنيات الأخرى لإنشاء أفضل صندوق أدوات لنمذجة النظم.

تكون تقنيات تطوير النظم نابضة بالحياة معظم الوقت، بخاصة النظم البرمجية. وهذا يعني أن أي منهجية موحّدة لنمذجة البرمجيات يجب

أن تكون مرنة ومفتوحة على المنهجيات الجديدة كي يستمر استعمالها عملياً؛ وعلى أية حال، يوجد بالنهاية مع لغة النمذجة الموحدة لغة مشتركة للتعبير عن النماذج.

المراجع

- Flanagan D., McLaughlin B., 2004. Java 5 Tiger: A Developer's Notebook, (1st Edition), O'Reilly.

- Freeman E., Freeman E., Sierra K., Bates B., 2004. Head First Design Patterns, (1st Edition), O'Reilly.

- Hamilton K. and Miles R., 2006. Learning UML 2.0, O'Reilly.

- Helm E., Johnson R., Vlissides R. and Gamma J.M., 2002. Design Patterns: Elements of Reusable Object-Oriented Software (3rd Edition) Addison-Wesley.

- David Flanagan, Java in a Nutshell, (3rd Edition), O'Reilly.

- Robert C. Martin, 2002. Agile Software Development, Prentice Hall.

- Scott Oaks, Henry Wong, 2004. Java Threads, (3rd Edition), O'Reilly.

ثبت المصطلحات

A

Abstract Classes	الأصناف المجرّدة
Abstraction	التجريد
Access specifier	محدد الوصول
Access to a package	الوصول إلى حزمة
Actions	الأفعال
Activation Bars	مستطيلات التنشيط
Activity Diagram	مخطط النشاط
Activity final node	عقدة نهاية نشاط
Activity Frame	إطار النشاط
Actor	ممثل، مستخدم
Administrator	مدير
Aggregation Relationship	علاقة التجميع
Agile Methods	الطرق الذكية
Alternative Notation	الترميز البديل
Ambiguity	الغموض
Annotations	الملاحظات
Anonymous Participant	مشارك مجهول الاسم

Argument	الوسيطة البيانية
Artifacts	الأدوات الاصطناعية
Assembly Connectors	روابط التجميع
Association	الشراكة
Association Classes	الأصناف بالشراكة
Asynchronous	غير المتزامنة
Attribute properties	ميزات الخاصية
Author Credentials Database	قاعدة بيانات الكتبة

B

Ball and Sockets Notation	ترميز الكرة والمقبس
Base Classes	الأصناف الأساسية
Binding Class Templates	ربط الأصناف القوالب
Black-Box Component View	منظور الصندوق الأسود للمكوّن
Blog	المدوّنة
Blog Entry	التدوينة
Blogger	مستخدم المدونة
Blueprint	ناسخ كربوني
Business Process Management (BPM)	إدارة عمليات الأعمال
Business to Business	شركة إلى شركة

C

Capture	أسر
Chain of Responsibility (COR)	سلسلة المسؤولية
Children Classes	الأصناف الأبناء
Class Attributes	خصائص الصنف
Class Behaviors	سلوكيات الصنف
Class Invariants	ثوابت الصنف، قيود ثابتة

Class Loader	محمّل الصنف
Classes	الأصناف
Classes Diagram	مخطط الأصناف
Collaboration	التعاون
Communication Diagram	مخطط الاتصال
Communication Lines	خطوط الاتصال
Component	المكوّن
Component-Oriented Software Development	تطوير البرامج المكوّنية التوجه
Components Diagram	مخطط المكوّنات
Composite property	الميزة تركيب
Composite State	حالة مركبة
Composite Structure Diagram	مخطط الهيكل المركّب
Composition Relationship	علاقة التركيب
Computer Aided Systems Engineering (CASE)	هندسة الأنظمة بمساعدة الحاسب
Concrete Classes	الأصناف الملموسة
Concurrence	التزامن
Configuration	الترتيبات، التهيئة
Confusion	الالتباس
Connectors	الروابط
Constraints	القيود
Constructor	المشيّد، الباني
Content Management System (CMS)	نظام إدارة المحتوى
Context Sensitive	المتأثر بالسياق

D

Debugging	التصحيح

Decision	قرار
Declaration	التصريح أو الإعلان عن
Default	الافتراضي، التلقائي
Delegation	التفويض
Delegation Connectors	روابط التفويض
Dependency	التبعية
Deployment Descriptions	مواصفات الانتشار
Deployment Diagram	مخطط الانتشار
Derived Classes	الأصناف المشتقة
Description	التوصيف
Design Patterns	أنماط التصميم
Development view	منظور التطوير
Doc-Style Tags	أوسمة وثائقية النمط
Drives	السواقات
Dropped Title Box Technique	تقنية صندوق العنوان المنخفض

E

Elements Visibility	رؤية العناصر
Embedded Systems	الأنظمة المغروسة أو المتضمنة
Encapsulation	التغليف، الكبسلة
Envent Handler	مدير الحدث
Event Listener	المستمع لحدث
Exact Time Measurements	مقاييس الوقت الدقيقة
Executable	قابلة التنفيذ
Expansion Regions	مناطق التوسّع
Expression Language	لغة التعابير
Extend Relationship	علاقة التوسّع

التوسيعات، الامتدادات	Extensions

F

التغذية الراجعة	Feedback
المخطط الانسيابي	Flowchart
الشوكات	Forks
الاستمارة	Form
لغات النمذجة الرسمية	Formal modeling language
اسم المدى الكامل	Fully-Scoped Name

G

علاقة التعميم	Generalization Relationship
المعمّمة	Generalized
التعميم	Generic
شرط الحراسة	Guard Condition

I

إعادة استعمال الشفرة	Implementation Reuse
استيراد حزمة	Import a package
لعلاقة التضمين	Include Relationship
الحالات المتضمنة	Included Cases
مسار قدوم	Incoming Edge
لغات النمذجة غير الرسمية	Informal modeling language
الوراثة	Inheritance
عقدة البداية	Initial node
ضمني، داخلي	Inline
مثيل الصنف، كائن	Instance
إنشاء مثيل	Instantiation

Interaction Overview Diagram	مخطط ملخّص التفاعل
Interfaces	الواجهات
Internal Structure	التركيب الداخلي
Internal Transitions	الانتقالات الداخلية
Interoperable	قابلة للعمل فيما بينها
Interrupting an activity	اعتراض نشاط
Interruption Regions	مناطق الاعتراض
Invariants	الثوابت
Italic Style	النمط الإيطالي للكتابة
Iterative Method	الطريقة التكرارية

J

Java Collection	المجموعة في جافا ، هيكل بياني
Joins	الموحدّات، الموصّلات

L

Library	المكتبة البرمجية
Lifecycle	دورة الحياة
List, Data Structure	القائمة ، هيكل بياني
Logger	محمّل السجلات
Logical Analyzer	محلل منطقي
Logical view	منظور منطقي
Login	الدخول للنظام

M

Macro	الماكرو
Map, Data Structure	الخريطة ، هيكل بياني
Message Arrows	أسهم الرسالة
Message Signature	توقيع الرسالة

Meta-model	نموذج النموذج
Method	الطريقة
Method body	جسم الطريقة
Model-Driven Architecture (MDA)	معمارية القيادة بالنموذج
Modeling System Workflows	نمذجة تدفقات عمل الأنظمة
Module	الوحدة
Multiplicity	التعدّدية
Multi-tier	متعدد الطبقات
Mutual exclusive	الإقصاء المتبادل

N

Namespace	فضاء الأسماء
Navigability	الصلاحية للملاحة
Nested Messages	الرسائل المتداخلة أو المتعششة
Node	العقدة
Not unique property	الميزة غير فريدة
Notation	الترميز
Notes	الملاحظات، التعليقات
null	القيمة المعدومة

O

Object Constraint Language	لغة قيود الكائن
Object Identity	هوية كائن
Object Modeling Group (OMG)	مجموعة النمذجة بالكائنات
Object Modeling Technique	تقنية النمذجة بالكائنات
Object Oriented	كائني التوجه
Object Oriented Analysis and Design (OOAD)	التحليل والتصميم كائني التوجه
Object Oriented Programming Language	لغة برمجة كائنية التوجه

Object Oriented Software Engineering (OOSE)	هندسة البرمجيات كائنية التوجه
Objects Diagram	مخطط الكائنات
Operation parameters	بارامترات أو وسيطات العملية
Operation return type	نوع إرجاع العملية
Operation Signature	توقيع العملية
Operation view	منظور العملية
Ordered property	ميزة الترتيب
Outgoing Edge	مسار خروج

P

Package Visibility	الرؤية الحُزمية
Packages Diagram	مخطط الحُزَم
Parallel	التوازي
Parent Classes	الأصناف الأهل
Participant Creation	إنشاء مشارك
Participant Destruction	هدم مشارك
Participant State-Line	خط حالة المشارك
Participants	المشاركون
Partitions	التجزئة ، الممرات
Path or Edge	المسار
Patterns	الأنماط
Physical view	المنظور المادي
Platform Independent Models (PIM)	النماذج المستقلة عن المنصّة
Platform Specific Models (PSM)	النماذج المحددة المنصّة
Ports	المنافذ
Postcondition	الشرط اللاحق
Precondition	الشرط المسبق

Primitive Data Type	نوع بيانات أساسي
Private Visibility	الرؤية الخاصة
Profiles	المظاهر، البروفايل
Programming Language	لغة برمجة
Protected Visibility	الرؤية المحمية
Protocol State Machine	بروتوكول حالة الآلة
Provided Interface	الواجهة المُجهَّزة
Pseudostate	شبه حالة
Public Visibility	الرؤية العامة

R

Read Only property	ميزة للقراءة فقط
Real Time Systems	الأنظمة الفورية
Realization, Implementation	الإنجاز، التحقيق، كتابة الشفرة
Really Simple Syndication (RSS)	خدمة متابعة الأخبار على الإنترنت
Receiving Signals	استقبال الإشارات
Recurrence	الدورية، الاستدعاء الذاتي
Redefine property	ميزة إعادة التعريف
Redundancy	الإسهاب، تكرار التخزين
Regular Expressions	التعابير المنتظمة
Relationships	العلاقات
Relative Time Indicators	مؤشرات الوقت النسبية
Remote Machine Invocation	اتصال عن بُعد بالآلة
Required Interface	الواجهة المُتطلّبة
Requirements	المتطلبات
Return Message	رسالة الرجوع

S

Scalable	متعددة المقاييس
Semantic	الدلالية
Sending Signals	إرسال الإشارات
Sequence Diagram	مخطط التتابع
Sequence Fragment Operators	عمليات قسم تتابع
Sequence Fragments	أقسام التتابع
Signals	الإشارات
Simplification	التبسيط
Singleton Design Pattern	نمط تصميم المثيل الوحيد
Sketch	التصميم الأولي
Software Development Process	عملية تطوير البرامج
Source File	الملف المصدر
Source State	حالة مصدر
Specialized	المخصّصة
Standard	المقياس، المعيار
State Internal Behavior	سلوك الحالة الداخلي
State Machine Diagram	مخطط حالة الآلة
State Transition Diagram	مخطط الحالة والانتقال
States	الحالات
Static	ساكن
Stereotypes	الحاشيات
String	سلسلة الرموز
Subclasses	الأصناف الفرعية
Subset property	ميزة المجموعة الفرعية
Substate	حالة فرعية

Subsystem	نظام فرعي
Super Classes	الأصناف العليا
Synchronous	المتزامن
System Boundaries	حدود النظام

T

Tagged values	القيَم الملحقة
Target State	حالة هدف
Technical Support Process	عملية دعم فني
Templates	القوالب
Threads	المسالك
Tightly couple	المقترنة بإحكام
Time Events	الأحداث الزمنية
Timing Constraint Format	بنية القيد الزمني
Timing Constraints	القيود الزمنية أو التوقيتية
Timing Diagram	مخطط التوقيت
Top-Level Sequence Diagram	مخطط تتابع عالي المستوى
Transition	الانتقال
Transition-Oriented View	المنظور الانتقالي التوجه
Trigger	المُطْلِق، البادئ

U

UML Tools Variations	اختلافات أدوات لغة النمذجة الموحدة
Unified Modeling Language (UML)	لغة النمذجة الموحدة
Union Property	ميزة الاتحاد
Unique property	ميزة الفرادة
Update	التحديث
Use Case view	منظور حالة الاستخدام

مخطط حالات الاستخدام	Use Cases Diagram
واجهة المستخدم	User Interface

V

الإسهاب	Verbosity
منظورات أو رؤى النماذج	Views of Models
نوع بيانات معدوم القِيَم	void

W

الطريقة الانحدارية أو الشلال	Waterfall Method
منظور الصندوق الأبيض للمكوّن	White-Box Component View

كشاف الموضوعات



السيرة الذاتية للمترجم

- نال الدكتور خالد سعيد خليل شهادة الدكتوراه في علوم الحاسب تخصص معالجة الصور من جامعة البورغون في فرنسا سنة ١٩٩٦م.
- ونال أيضاً شهادة الماجستير تخصص هندسة البرمجيات من المؤسسة الوطنية للعلوم التطبيقية في ليون بفرنسا INSA de Lyon سنة ١٩٩٢م.
- كما نال شهادة بكالوريوس علوم الحاسب من جامعة ليل للعلوم والتكنولوجيا في فرنسا سنة ١٩٩١.
- قام الدكتور خالد خليل بالتدريس كمحاضر في جامعة البروغون طيلة ثلاث سنوات خلال فترة إعداد الدكتوراه.
- رجع إلى وطنه لبنان بعد ذلك وقام بالتدريس مدة ثلاث سنوات في أعرق الجامعات اللبنانية.
- انتقل إلى جامعة الملك فيصل حيث يعمل فيها كأستاذ مساعد منذ أيلول ٢٠٠٠م في كلية إدارة الأعمال وسابقاً في كلية علوم الحاسب وتقنية المعلومات.
- لقد قام الدكتور خالد خليل بتدريس معظم مواد علوم الحاسب من برمجة وتحليل وتصميم وقواعد بيانات وشبكات وتقنية المعلومات.
- يهتم الدكتور خالد خليل بالمجالات البحثية: التحليل والتصميم والبرمجة كائنية التوجه، التنقيب في البيانات واستخراج المعرفة، ومعالجة الصور بشكل عام.



مطابع جامعة الملك فيصل
SALEH
WWW.KFU.EDU.SA

ردمك : ٩٧٨-٩٩٦٠-٠٨-٠٩٧-٠